$2.95 USA

Australia	A $ 4.75	New Zealand	NZ $ 6.50
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

'68' MICRO JOURNAL

VOLUME VII ISSUE II • Devoted to the 68XX User • February 1985

"Small Computers Doing Big Things"

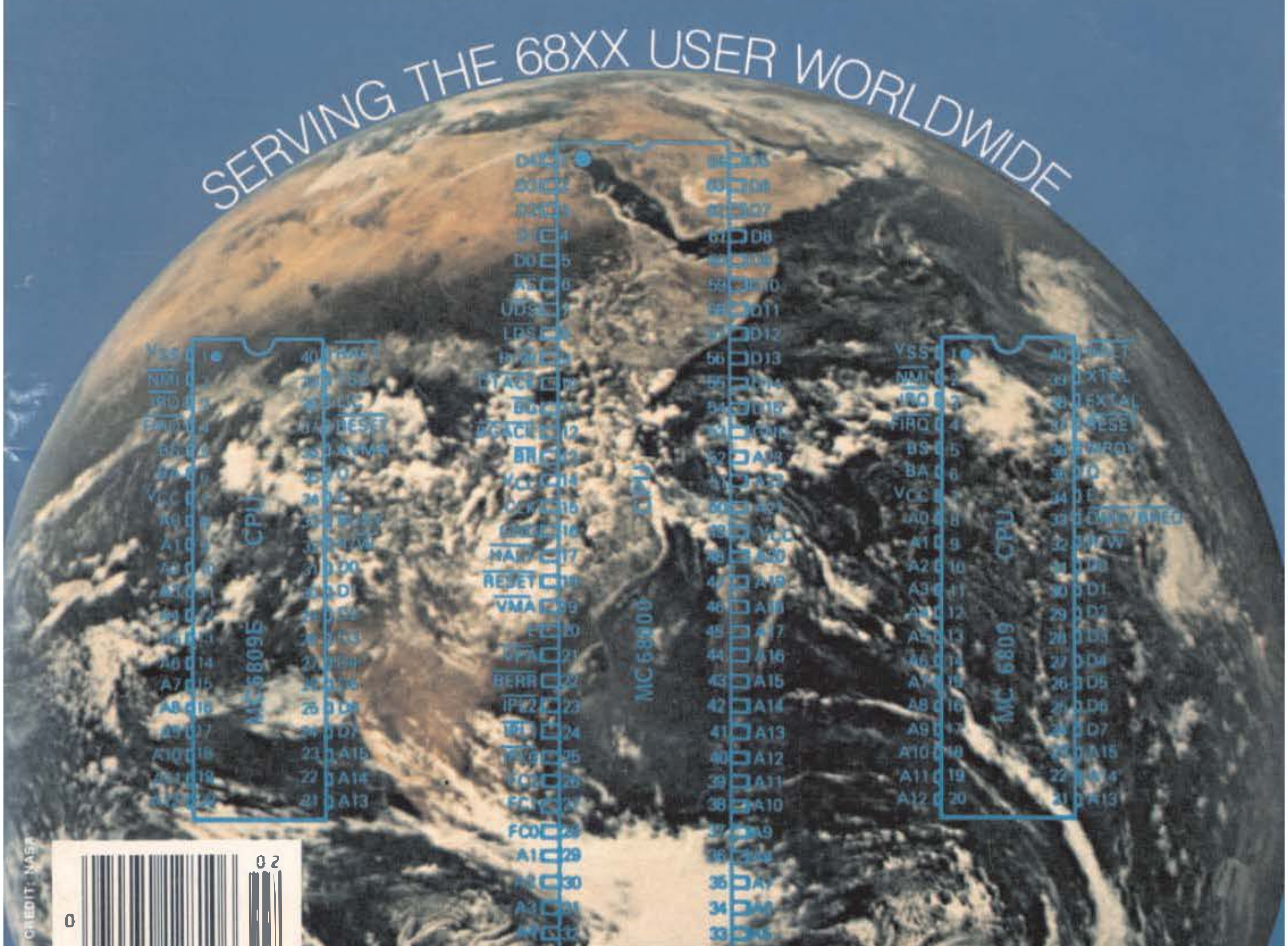
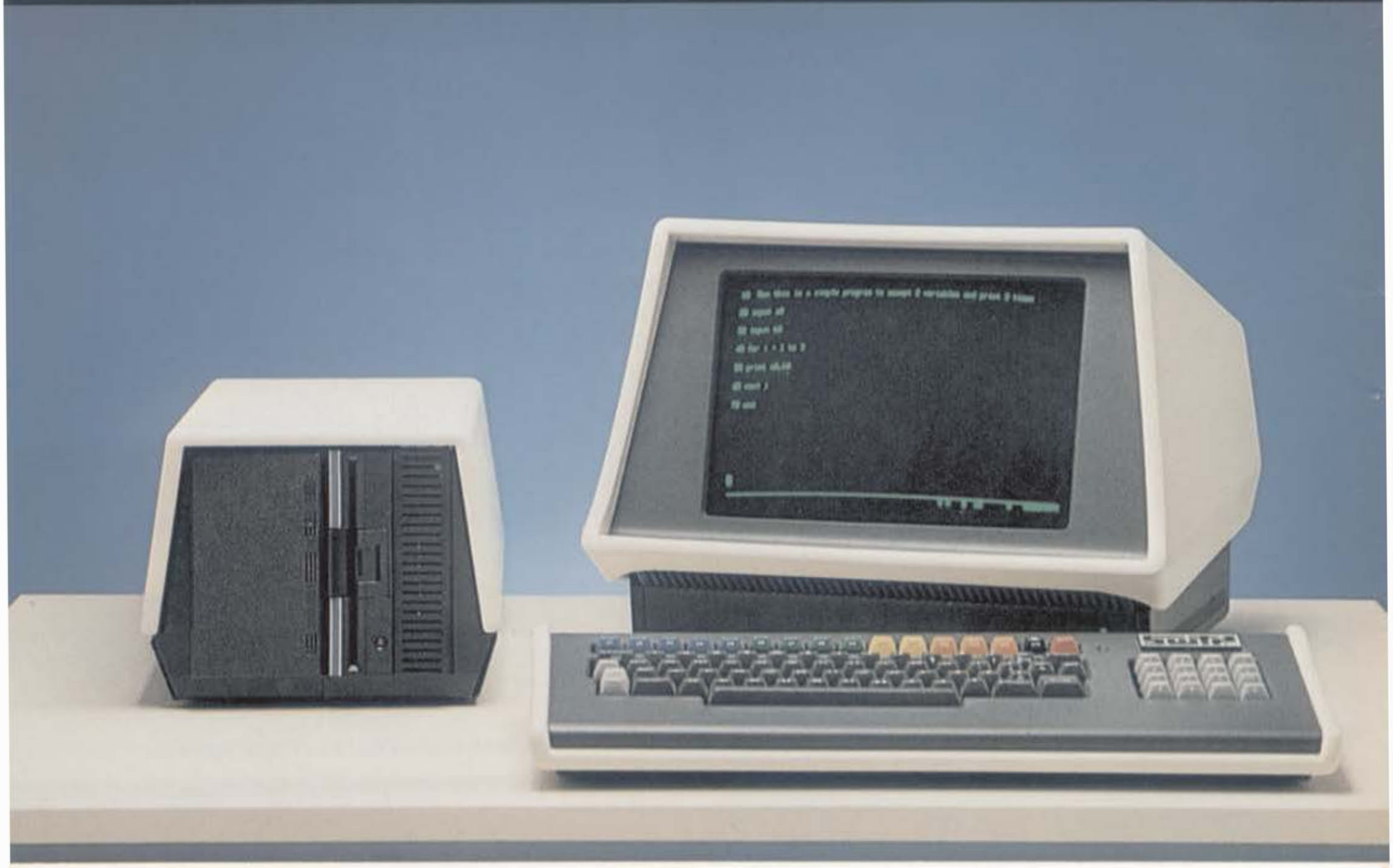


PHOTO CREDIT - NASA

WE DON'T PLAY GAMES



X-12+ A SERIOUS COMPUTER IN A DESKTOP PACKAGE

Multiprocessor Technology - Combination of 8, 16 and 32 bit types

1.0 Megabyte Memory - Insures no limitation on programs

"Winchester" Disk System - Fast response, large storage capacity

UniFlex* Operating System - The standard of comparison

Hardware Floating Point - Unmatched speed in a small system

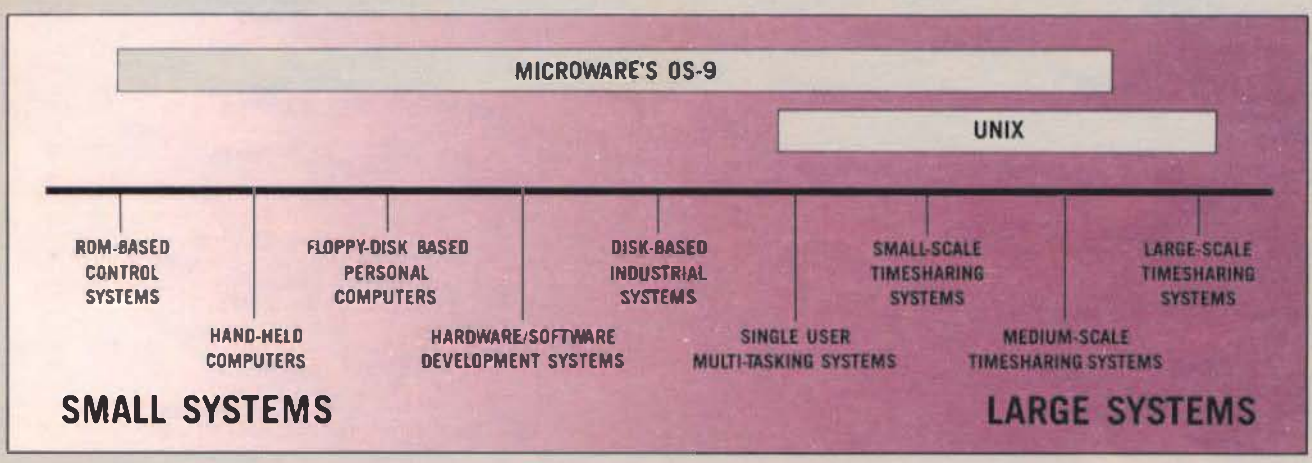
Up to Three Terminals - Instant expansion

*Trademark of Technical Systems Consultants

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum


MICROWARE'S OS-9
UNIX
ROM-BASED CONTROL SYSTEMS
HAND-HELD COMPUTERS
FLOPPY-DISK BASED PERSONAL COMPUTERS
HARDWARE/SOFTWARE DEVELOPMENT SYSTEMS
DISK-BASED INDUSTRIAL SYSTEMS
SINGLE USER MULTI-TASKING SYSTEMS
SMALL-SCALE TIMESHARING SYSTEMS
MEDIUM-SCALE TIMESHARING SYSTEMS
LARGE-SCALE TIMESHARING SYSTEMS
SMALL SYSTEMS
LARGE SYSTEMS

Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

microware®
OS-9™

MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
Phone 515-224-1929
Telex 910-520-2535

Microware Japan, Ltd
3-8-9 Baraki, Ichikawa City
Chiba 272-01, Japan
Phone 0473(28)4493
Telex 299-3122

OS-9 is a trademark of Microware and Motorola. Unix is a trademark of Bell Labs.

'68' MICRO JOURNAL

Portions of the text for 68 MICRO JOURNAL was prepared using the following furnished hard/software.

COMPUTERS-HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216
S09-5/8 DMF disk-CDS1-8212W-Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe-OS9-FLEX-Assorted Hardware

EDITORS-WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX-Editor-Processor

reat Plains Computer Company, Inc.
PO Box 916
Idaho Falls, ID 83401
STYLO-Mail Merge

Editorial Staff

Don Williams Sr.	Publisher
Larry E. Williams	Executive Editor
Tom E. Williams	Production Editor
Robert (Bob) Nay	Color Editor

Administrative Staff

Mary Robertson	Office Manager
Penny Williams	Subscriptions
Michael Westfall	Shipping/Rec.
Christine Kocher	Accounting

Contributing Editors

Ron Anderson
Norm Commo
Peter Dibble
Dr. Theo Elbert
William E. Fisher
Dr. E.M. Pass

Special Technical Projects

Clay Abrams K6AEP
Tom Hunt

CONTENTS

Vol.VII,Issue II February 85

FLEX USER Notes.................... 7 Anderson
OS9 USER Notes..................... 9 Dibble
C USER Notes....................... 12 Pass
68000 USER Notes................... 15 Lucido
Software Tools in PASCAL........... 17 Bollinger
Turbo.............................. 24 Groves
Single Board Computer.............. 26 DMW
MICROKEY 4500...................... 27 Dale
Local.............................. 29 Armstrong
TSC BASIC To Microsoft BASIC....... 34 Pass
Bit Bucket......................... 42
Classified Advertising............. 52

Send All Correspondence To:

Computer Publishing Center
68 MICRO JOURNAL
5900 Cassandra Smith
PO Box 849
Hixson, TN 37343
Ph (615)842-4600 TELEX 558 414 PVT BTH

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tenn. and additional entries. Postmaster: send Form 3579 to 68' Micro Journal, PO Box 849, Hixson, Tennessee.

SUBSCRIPTION RATES
USA
1-Year $24.50 2-Years $42.50 3-Years $64.50
FOREIGN
See Page 60

Items Submitted for Publication

Articles submitted for publication should be accompanied by the authors full **name, address, date and telephone number.** It is preferred that articles be submitted on either 5 or 8 inch diskette in TSC Editor format or STYLO format. All diskettes will be returned.

The following TSC Text Processor commands ONLY should be used (due to our proportional processor): .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. The rest we will enter at time of editing.

STYLO commands are all acceptable except the .pg page command, we print edited text files in continous text.

All articles submitted on diskettes should be in TSC FLEX™ format, either FLEX2 6800, or FLEX9 6809 any version.

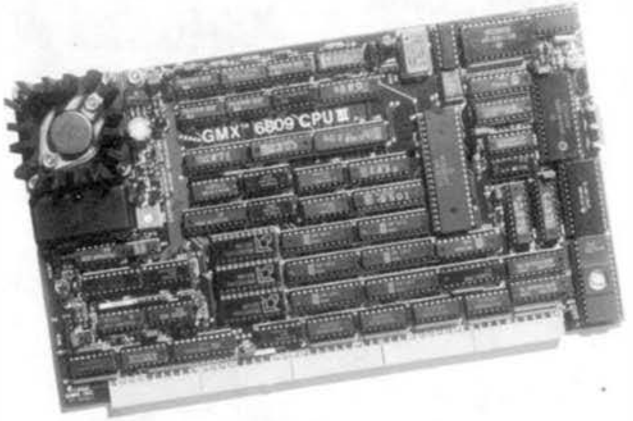
If articles are submitted on paper they should be on white 8X11 bond or better grade paper. No hand written articles (hand written or drawn art accepted). All paper submitted articles will be photo reproduced. This requires that they be typed or produced with a dark ribbon (no blue), single spaced and type font no smaller than 'elite' or 12 pitch. Typed text should be approximately 7 inches wide (will be reduced to column width of 3 1/2 inches). **Please use a dark ribbon!**

All letters to the editor should also comply with the above and bear a signature. Letters of 'gripes' as well as 'praise' are solicited. We attempt to publish all letters to the editor verbatim, however, we reserve the right to reject any submission for lack of 'good taste'. We reserve the right to define what constitutes 'good taste'.

Advertising: Commercial advertisers please contact the 68 Micro Journal advertising department for current rate sheet and requirements.

Classified: All classified must be non-commercial. Maximum 20 words per classified ad. Those consisting of more than 20 words should be figured at .35 cents per word. 20 words or less $7.50 minimum, one time, paid in advance. No classified ads accepted by telephone.

GIMIX STATE OF THE ART 6809 SYSTEMS FOR THE SERIOUS USER.



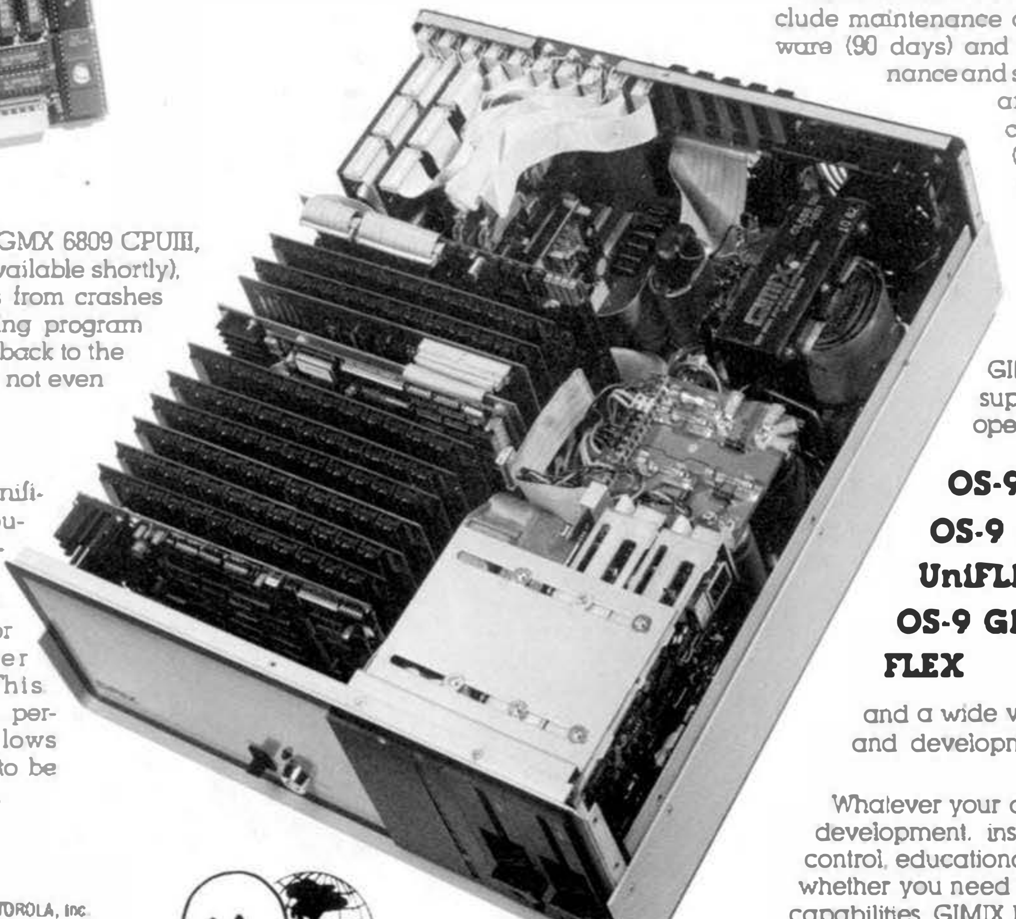
GIMIX has 19MB or high performance 47MB Winchester Drive Systems and/or Floppy Disk Drive Systems.

For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer.

GIMIX has single user systems that can run both FLEX and OS-9 or Multi user systems for use with UniFLEX or OS-9.

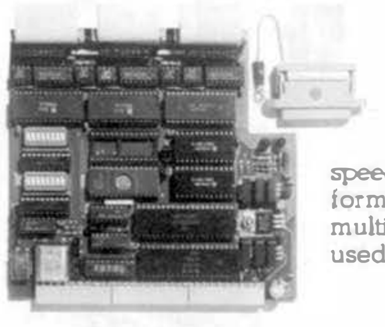
GIMIX versions of OS9 and UniFLEX include maintenance and support by Microware (90 days) and TSC (1 year). Maintenance and support after this period are available at extra cost.

(NOTE: this support and maintenance is only for use with approved GIMIX hardware)



For the ultimate in performance, the Unique GMX 6809 CPUIII, using either OS-9-GMXIII or UniFLEX GMXIII (available shortly), gives protection to the system and other users from crashes caused by defective user programs. e.g. During program development, a programmer who crashes goes back to the shell or the debugger, while the other users are not even aware anything occurred.

The intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions, thereby freeing up the host CPU for running user programs. This speeds up system performance and allows multiple terminals to be used at 19.2K baud.



GIMIX 6809 systems support five predominant operating systems:

OS-9 GMX III,
OS-9 GMX II,
UniFLEX,
OS-9 GMX I,
FLEX

and a wide variety of languages and development software.

Whatever your application: software development, instrumentation, process control, educational, scientific or business, whether you need single or multi-user capabilities, GIMIX has hardware and the operating systems to get the job done reliably.

Please phone or write if you need further information.

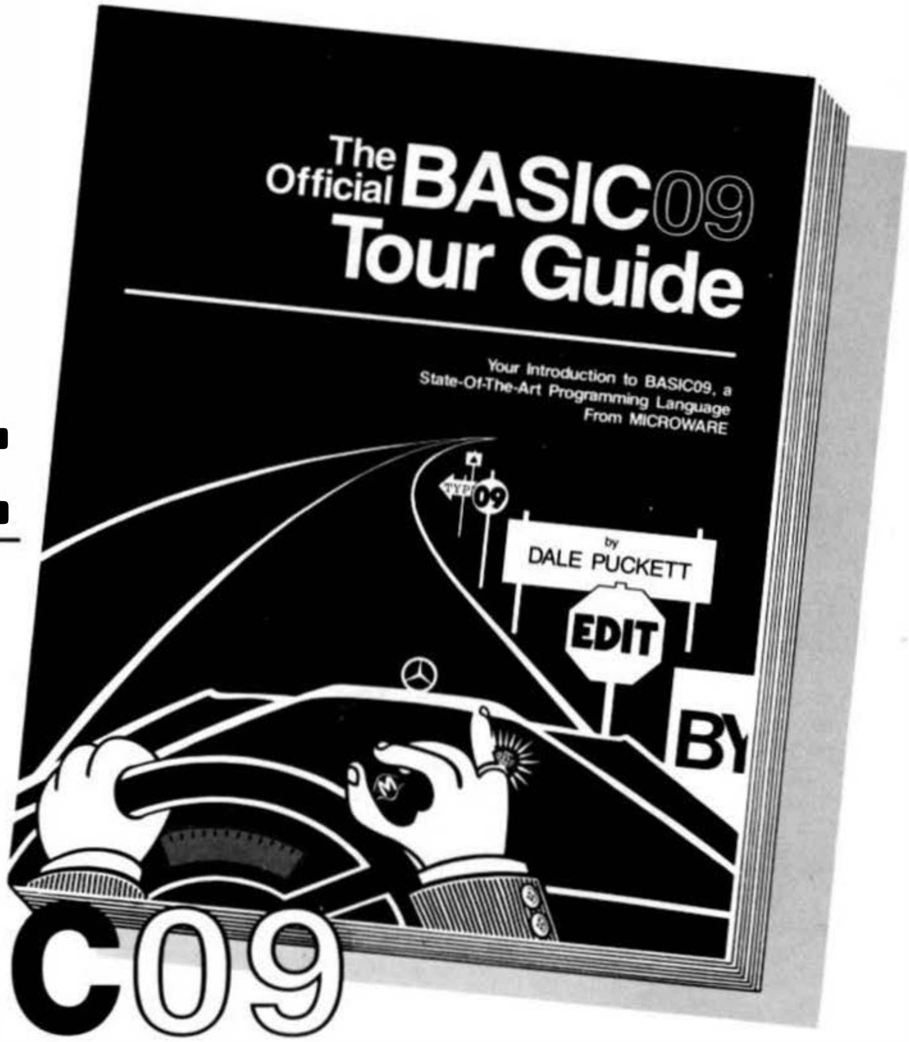
BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc.
FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc.
GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

GIMIX inc.

1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

'68' Micro Journal

Get the most out of BASIC09


The Official BASIC09 Tour Guide
Your Introduction to BASIC09, a State-Of-The-Art Programming Language From MICROWARE
by DALE PUCKETT
EDIT
09
BY

The **OFFICIAL BASIC09 TOUR GUIDE** is skillfully written in a friendly and easy-to-read style. Just perfect for those new to computers and to BASIC09. It's also a *valuable reference book* for programmers, engineers, students and hobbyists, providing an in-depth look at BASIC09 plus an overview of the OS-9 operating system. Comprehensive reference sections on BASIC09 and OS-9 commands are also included.

The book "maps" out your route through the Mercedes of Basics . . . BASIC09 and puts you in the driver's seat in no time. Fasten your seatbelt, sit back and enjoy the ride to perfecting your programming skills.

MICROWARE . . .

The **OFFICIAL BASIC09 TOUR GUIDE** comes from the people who wrote BASIC09. As the leader in 6809 system software, we at MICROWARE care about our users and want to help you get the most from our products.

It's Easy to Order.

Phone orders are accepted from MasterCard or VISA cardholders or for COD shipment. You can also order by mail using the coupon below. Quantity discounts are available to educational organizations and dealers. For further information contact Microware.

microware®

Specialists in system software for 68-family microprocessors since 1977.

OS-9 and BASIC09 are trademarks of Microware and Motorola.

Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322
Telephone 515/224-1929
Telex 910-520-2535

Please send ________ copies of the **Basic09 Tour Guide** book at $18.95 each. Add $2.00 for UPS shipping in the U.S. or $5.00 for overseas air mail per book. Iowa residents add 4% sales tax.

Name ____________________

Address ____________________

City ____________________

State __________ Zip __________

□ I have enclosed a check

□ Charge to my bank card:

MasterCard □ VISA □

Card Number ____________________

Expiration ____________________

More help than any other thing
I have ever read, except 68 Micro
Journal, of course. Come on with
the C book on Norm Commo's column.
BD

FLEX™ USER NOTES

THE 6800-6809 BOOK

SWELL! - Best buy EVER in a
6800/6809 book.
JTW

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES,** in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.

Finally I don't have to type in all
that source code, a GREAT, GREAT idea.
Thanks and good luck, you guys are the
best!
TOA

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

I will keep this (FLEX USER NOTES) book
always beside, my computer. I have not
ever, with any magazine, received as much
information as this one book, and 68 MJ.
DIB

See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

TELEX 558 414 PVT BTH

™FLEX is a trademark of Technical Systems Consultants

SOFTWARE DEVELOPERS!

YOU'VE JUST BEEN GIVEN THE BEST REASON YET
TO GET OUR 68000/UNIX® DEVELOPMENT SYSTEM

THE VAR/68K® SERIES

VK-5XW20 (List price ~~$10,100.~~) **$5,000.**
Includes: Terminal, 20 Mb hard disk, 512K RAM, 8 ports and REGULUS®

VK-5XW20T20 (List price ~~$12,900.~~) **$6,500.**
Includes: all of above, plus 20 Mb tape streamer

RESELLERS!

Even more attractive specials are available to qualified resellers!

Smoke Signal has been designing, developing and manufacturing microcomputers based on the Motorola family of processors for the past six years. The VAR/68K is the most recent addition to our family of multi-user computers.

VAR/68K is a registered trademark of Smoke Signal
REGULUS is a registered trademark of Alcyon Corp.
UNIX is a registered trademark of Bell Laboratories

Due to the extremely low prices being offered, we can only accept cash or C.O.D. orders, and we must limit purchases to one per customer. This is a limited time offer.

**TO OBTAIN YOUR VAR/68K
AT THESE LOW PRICES, CONTACT:**

SMOKE SIGNAL
31336 VIA COLINAS
WESTLAKE VILLAGE, CA 913 2
(818) 889-9340 / Telex 910-494-4965

Flex User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

An Editorial of Sorts

You know, I've been giving a lot of thought lately to the subject of progress. Just what is the most effective hardware to do an industrial control or instrumentation job. We all know that new things cost a great deal at first. Therefore it is reasonable that if I don't need the extra capabilities of new processors, I would be foolish to pay for them. On the other hand, it is possible to argue that "I have a working design. Why change it?" to the point of becoming very obsolescent. For example, there are still 6800 processors being built into instruments. If they have all the capabilities, and the software is done once and for all, that is all just fine. Why not keep building the instrument or whatever just as it is now.

Now here is where I think the fallacy lies. If the product requires a great deal of software support, perhaps a different program for each customer, maybe the designers are just in a rut. Think for a moment about the superior instruction set of the 6809 as compared to the 6800. Any reasonably good assembler programmer can generate the same program for the 6809 with 15% to 30% fewer lines of instructions. I believe that the difficulty of debugging any program increases faster than the number of lines of code. Perhaps a good estimate would be that the debug time is about proportional to the square of the size of the program. By that estimate, a program 30% smaller will take half as long to debug. Also, it will take less time to write and to list. Another factor to be considered is that the 6800 and 6809 are very similar and minimal circuit changes are required to make the conversion. Hardly any other component changes need be made, so that the cost of switching is minimal, and the cost per unit is also increased very nominally. I therefore argue that anyone who has a 6800 based product that requires program adaptation or rewriting should seriously consider switching.

When it comes to the use of the 68000 however, (again my argument is limited to machine control and instrumentation applications), the situation is somewhat different. The 68000 costs several times as much as the 6809, and it requires more and faster hardware to support it. Bus speeds are higher, making noise immunity lower in industrial environments. In general the premium paid for using "the latest" is not small. I've been saying for a couple of years now, that as soon as I had an application the 6809 couldn't handle, I would use a 68000. I'm still waiting for that application, and I still have plenty of room to improve my 6809 handling of applications. Presently, my designs are still running the 6809 at 1 MHz. I can do everything twice as fast by plugging in a 68B09, an 8 MHz crystal, and the B version of the serial and parallel interface chips. Memory is already capable of 2 MHz.

I'm not saying that tomorrow won't bring me a problem that requires the extra speed of an optimized 68000 assembler code solution, just that such a problem has not yet come forth. I think many designers don't try very hard to squeeze some extra performance out of their present hardware. I can relate tales of improving performance (execution time) by using better software, by a factor of about 200, and that on a 6800 system. I think the early 6809 software that just echoed the 6800 code didn't take advantage of the 6809. In the past few years, improvements of a couple orders of magnitude have taken place in the performance of the object code generated by 6809 compilers. (Language is irrelevant to this discussion). The newest of the compiler implementations are still showing single digit percentage gains in performance, but the gigantic improvements have pretty much taken place.

Benchmarks run on the 68000 early on, were very disappointing. It usually turned out that a 2 MHz 6809 could match or exceed the performance of the 68000 on the same benchmark. However, I think the same thing has happened with the 68000 as happened early with the 6809. Programmers did not immediately learn to take full advantage of the 68000 instruction set. More recent results indicate that the 68000 can do things a good deal faster than the 6809 if its resources are used wisely, (such as the use of registers to hold variables). Yes, the 68000 is coming of age.

Now before several hundred of you start writing me letters about my stupidity, let me repeat in no uncertain terms that I AM NOT TALKING ABOUT ALL APPLICATIONS HERE. I don't know how to shout my point in writing, but if I did, I would. No, I am NOT talking about trying to misapply a 6809 to a 12 user super office computer, a CAD system, the control of a complex robot or a multi-axis CNC machine tool application. I am ONLY talking about applications in reasonably simple controls and instrumentation. I am talking about "canned software" in a Stand Alone system with very limited or no user programmability beyond the input of "set up" parameters. I'm talking about what I call a "dedicated computer" application. In such applications, there is generally NO mass storage device. There MAY be some battery backed up RAM to allow the system to remember certain constants and limits while power is off. Generally the program is in ROM.

I hope this will have sufficiently narrowed down the applications about which I am speaking. I think most of the differences of opinions that have been aired in this column come about because we each tend to see something entirely different when someone says "Computer". We see a computer in the configuration in which we use it ourselves. Many of us are not even aware of all the applications that have become feasible both technically and economically because of the existence of the microprocessor and the ever tumbling prices for its peripheral devices. This difference in perception of what a computer is, is after all partly because a computer is a VERY general purpose tool. My little development system is nothing at all without software. At the moment it is a very nice text editor - processor. In a little while, it will be a software development system as I translate some of the software modules that I use frequently into a new language that I am studying. At other times it is a design tool that lets me explore alternate ways of solving a problem, at times through simulation techniques. Sometimes it is a tool to do a plodding search of several thousand possibilities to find the best solution to a specific problem. I have several friends who own computers, and for them, the word brings forth entirely different visions of what the computer is. For one, it is a data collector for experiments in Chemistry. For another it is a data management system for student grades, a word processor for the preparation of class notes and quizes, writing of technical papers, books, letters, etc. Your concept of what a computer is, may not agree with any of those I've mentioned here. Perhaps for you, computer means a system to handle you company's payroll and accounting records. If you work for an airline, your idea of a computer is that it can store and instantly recall vast amounts of information about flights, passengers, seating arrangements, prices, timetables, etc. I think the point is made, so I will stop here.

We Engineers have to consider several factors in designing a system. One of the major factors is cost. The latest, best, and newest technology, the "state of the art" things are always very expensive at first. Once the bugs in the production of these new items are worked out, and the design costs more or less paid for, the price nearly always takes a large drop. Somewhere along the way, the price begins to level off, and that is usually the point where it becomes economically feasible to use the new technology. Suppose I am using an EPROM that stores 4K bytes. It costs $4 in some quantity. A new device stores 16K bytes. It costs $64 in somewhat smaller quantity.

First of all, if my requirements for storage are around 4K or 8K, the new part may never become more economical. On the other hand, if I need 16K or 32K, the new part will become economically feasible before the price per K of storage is the same as the old device. I need only 1/4 as many of these devices. Handling is reduced. Programming time is reduced because I don't have to handle four devices, just one. Printed circuit board space is reduced, resulting in savings. The board only needs one socket, probably with a few more pins than the original device for which I need four sockets to get the same storage.

I've tried always, to work down at that point where the price of new devices starts to level off after the initial high price phase. I think that maximizes the value of the design. Of course, with the rapid changes in technology over the past decade or so, today's maximum value design is not tomorrow's. A designer must keep up with the latest items constantly, and a design can't stay static for very many years in most cases. If it does, you can be sure that the competitors will soon have something that does more, is simpler, and costs less. Enough said on this subject.

Computer Bargains

A few words for you out there with the limited budgets (that certainly includes most or all of us). Recently, I've acquired a couple of used SS-50 systems for very reasonable prices. I was fortunate to find a couple of sellers who realize that an original SWTPc 4K memory board is not worth anything. (You would need 14 of them to have 56K of memory). An 8K board is not useless, but is certainly of limited value. Old 6800 processor boards such as the MP-A and later MP-A2 are certainly of little value to most of us who want 6809 systems. And lastly, a pair of old 35 track single sided disk drives with a disk controller that won't run double density, no matter how little they have been used, and how reliable they are, are not worth a great deal, since now anyone who reads the ads in the magazines can pick up a double sided double density 40 track drive for less than $150. One of these will hold just about 370K bytes of data. Though the old 35 track drives cost nearly $1000 for a pair in a box with a power supply, you can now, with a little ingenuity put two drives and a power supply together for around $400, and have 700K of storage. How much is the 35 track single sided drive worth that holds 92K of data? It can't be worth more than $50 or $75.

My point is that these facts are not all bad. If you want to get into computing at minimal cost, find someone with an old SWTPc box containing a 6800 or 6809 processor board, 32K or more of memory, and a few I/O ports, a disk interface and a pair of drives, and you can be in business for a very small investment. You simply have to realize and accept the fact that you don't have the latest, fastest, largest system. Most sellers of these old SWTPc systems are selling because they are going into a more modern computer such as a Macintosh or an IBM PC for the simple reason that there is a great deal of software for these systems. Such sellers usually are willing to sell their original software at bargain prices too. I recently picked up two spreadsheet programs and a database program for the 6809 as part of a purchase.

As I've said before, the beauty of a "component system" or a bus system if you like, is that it can be upgraded a step at a time. You can replace those 4K memory boards with used 8K boards one at a time until you have 56K installed. You can now buy 64K boards new for around $200, and reduce the memory board count to one. If you become affluent later, you can buy a 256K board that takes less power than some of the old 8K and 16K boards. You can upgrade disk controllers and drives one at a time until you have a very capable system. Meanwhile you can be learning about computing and software as your system grows, and you never have to throw away anything of great value to go to the "next step up" in your computer.

You say "yes but I can't afford a terminal!" Just look around and be patient. I've lately seen a couple of perfectly good and serviceable terminals for $250 each. Find someone else who is upgrading from an old terminal to something more up to date, and take advantage of his upgrade to get yourself a terminal. Is there a junior college nearby? Schools sometimes upgrade the systems installed for student use. Perhaps they have a dozen terminals for sale and you can be an early customer and get the pick of the lot for the same price as the worst.

"Gee" you say, that still adds up to lots of money compared to a Commodore 64 or something like that. Of course you are right. The point is that you don't have to spend all that money at one time, or even in one year. I suppose I could look at the system I have right here and conclude that it is the most expensive electric typewriter that I could buy.

In my case, I have made enough on consulting fees over the years, and on articles that I have written for magazines in the past, to pay for my system a couple times over. What I have learned through what started out as a hobby interest has brought about a job for me in a highly paid position doing work that I thoroughly enjoy. If I look at my computing equipment as an educational investment, I've spent far less than it would cost to go to college for a couple of years at today's prices. The fact that you are reading this indicates that you have more than a passing interest in computers. You didn't buy a Color Computer and stop at playing games on it. It is not gathering dust as a doorstop somewhere in your house. If you find computing to be exciting, you have some success in writing programs and/or designing and building computer hardware, consider a computer an investment in your future, and work toward the goal of a career in some area of computing.

Cobol

I recently reviewed Crunch Cobol in this publication. I'm really glad that I took the approach that I did, and that I didn't claim to have written an elegant program in Cobol as the example.

I've just read the reply from Compusense that illustrates a couple of better ways to write that program in Cobol. As I said in the little review, I had expected someone to show me that I had done the program the hard way, and I appreciated the kind words in that reply. (They didn't even call me stupid!!)

Of course, the good solution to the problem involves using the REDEFINES feature of Cobol. The folks at Compusense wrote a letter to '68' Micro Journal (see Ed's Notes), and I think Don will publish their lesson on Cobol that shows how to redefine a character in working storage as an integer, perform the operation of converting it and then recover it as a character to go into the output string. I won't go into trying to describe their program since they do a fine job of that in their reply. I also received a letter from Mike Martin of Weatherford, TX, also a Cobol programmer who indicates that the Compusense implementation "is a good deal for a hundred bucks". He also sent me a solution to the case conversion program that uses the REDEFINES feature of Cobol. Thanks to both Compusense and Mike for setting us all straight on one of the most useful features of Cobol. Mike also indicates that a very good book on Cobol is "A Guide to Structured COBOL With Efficiency Techniques and Special Algorithms" by Pacifico A. Lim (Van Nostrand Reinhold Company). Mike goes on to say that most books on Cobol lean toward the IBM version, and that a number of features of Cobol discussed in these books are not supported by Crunch Cobol. Since Don will probably print Mike's letter and program in its entirety, I won't dwell on this any further. Thanks to both of you for setting the record straight.

Long time readers might remember a similar occurrence when I first started writing a few programs in "C". I had done a memory dump in Ascii and HEX program in a couple languages and tried a "C" version. Norm Commo was quick to show me a better way to write that program in "C", and I appreciated the lesson. I can say the same for the lesson on Cobol from Compusense. While I am too much a Pascal and "C" programmer to agree that having only "static" or "global" variables to work with is better, I do see a clarity in Cobol. I can readily understand how a good Cobol programmer would have little trouble maintaining someone else's code, and I appreciate the lesson, though I still don't see that "ADD A TO B GIVING C" is clearer than C := A+B;.

- - -

Ed's Notes: Gotcha, Ron. Beatcha to the draw. See January 1985 68 Micro Journal, BIT BUCKET section.

DMW

- - -

OS9 USER NOTES

By: Peter Dibble
517 Goler House
Rochester, NY 14620

Automated Updates

This month's column will be relatively short. I have a program I want to include that's longer than the usual, but I think it's worth sacrificing some text for.

I keep mentioning that I've turned into a full-time graduate student. The effects are finally beginning to show (bleary eyes from no sleep and lots of staring at books and terminals). I've spent a lot of time using UNIX. Two program development tools seem especially useful and much needed in OS-9. I wish I could write a debugger like DBX and include it in this column, but that's beyond me. Make, however, is a program I can fake.

Make is a UNIX program that looks at first like a version of the shell with a few special features that make it especially suited to running sequences of programs that make something. In the simplest case it is like packaging a long cc ... command line in a command file to save yourself from having to retype it every time you compile the program. Once I bothered to look into it I discovered that Make is much more that a special shell. The most important part of Make is its ability to understand dependencies.

A complicated program is composed of many pieces. There are a number of separately compiled modules with each module requiring one or more source files. If any of the sources have been changed since their modules were last compiled, they need to be recompiled. If any of the modules have been updated since the program was last linked, the module needs to be relinked. If you construct systems of programs, a modification to one of the programs may result in regeneration of some composite files -- maybe you'll want to print a new manual.

Make automates all this. You build a file that details the dependencies (prerequisites) for each file that is generated as part of the construction of the program. It also contains the command line that generates each file. Make checks the last-modified date and time on the file it's generating and each of the files that it depends on. If any of the dependencies are more recent than the file it's making, Make runs the command to build a new file. The program is arranged so it checks each dependency to see if it should be updated before it uses it to decide whether to update the final result.

An example would probably make this much clearer. Say you have a terminal simulator. It is divided into four modules: Setup, Run, Transfer, and Print. This would be stated (to my version of Make) as:

```
 /h0/cmds/terminal: Setup.o Run.o Transfer.o Print.o
 =cc2 Setup.o Run.o Transfer.o Print.o -f=terminal
```

That is: If any of Setup.o, Run.o, Transfer.o or Print.o have been updated since /h0/cmds/terminal, run the command line starting with cc2.

Make also understands that the .o files may need to be updated. The descriptions for them might be something like:

```
   Setup.o: Setup.a ACIA.Codes.h /h0/defs/OS9Defs
   Menu.fcbs CursorControl.a
   =cc2 Setup.a -o

   Run.o: Run.c ../localdefs/terminal.h
   /h0/defs/stdio.h
   =cc2 Run.c -o
   Transfer.o: Transfer
   =rma transfer -o=Transfer.o

   Print.o: Print.c /h0/defs/stdio.h
   =cc2 Print.c -o
```

When you run make against this file, it will first check Setup.o. If any of the dependencies for Setup.o (there are five of them) have been modified since Setup.a was last assembled Make will reassemble it. Then it will check run.o, transfer.o, and print.o in the same way. Finally make will come to /h0/cmds/terminal. If any of its dependencies have been modified since terminal was last modified, it will be relinked. This applies even if the dependency was updated by an earlier step in this make.

Often the depth of nesting goes beyond two. You'll be working on a system that includes files that depend on files that depend on other files and so forth. Make can deal with any degree of complexity. The only limitations are artificial. I set the constant DEPENDENCIES. You may reset it to a larger number if you need to. The C compiler generates the other limitation by chosing a default memory allocation. If you want to nest dependencies very deeply, it would be good to give make some extra memory. The procedure, resolve, allocates extra stack space for each level of nesting.

Documentation

The version of Make with this column was written in Microware C. I don't think I used any strange features so it should be easy to convert to other versions of C.

Make isn't any good without a "makefile." You'll have to write one up for every program (or system) you want to use Make with. The convention is to put all the files associated with a program in a directory by themselves and call the makefile for that program "makefile". If you just run Make, it will look for a file called makefile in the current data directory.

If you don't want to call your makefile "makefile," or you want to keep several in a directory, you can tell Make to use a different name for the makefile by putting the name on the command line:

```
OS9:make prog.d
```

would use a file called prog.d as the makefile.

The first line(s) of the makefile must contain the dependencies of the highest level file -- the end result of the make -- with the command that generates the top-level file next. From then on the files can be specified in any order. First a dependency line denoted by the name of the file it's referring to, followed by a colon and the list of dependencies; then an equal sign and the command line for that file.

The command lines are restricted to 80 characters, but the dependency lines can be any length or be several lines long.

As usual for programs in this column there is a lot that needs improving in this program. It is good enough to be very useful, but there's plenty of room for bells and whistles. If I still feel interested in it next month, I may cook up a fancy version over Christmas break and see if I can get Don to sell it for me.

I Spoke Too Soon

Just before the Microware Seminar I heard from Gimix that they had stopped work on their 68000 board. I was very unhappy about it, but hoped they might change their mind. Last month I gave up and groaned about it a little in this column. Just a few weeks after I sent the column in, I heard that Gimix was working on the board again. Now, it's true that the board is being designed to work with UniFlex, not OS-9, but at least there is hope. Remembering the number of Gimix CPUs in Microware's lab I think there is good reason to hope for OS-9 support sometime next year. I don't know much about the board, but from what I do know combined with my experience with UNIX, I'm looking forward to showing my Computer Science friends my micro running about as fast as their minicomputers.

- - -

```
#include <stdio.h>
#include <direct.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0
#define DEPENDENCIES 20
static char *Makefile = "makefile";
FILE *f;
static struct
 {
     char *Name;
     char ModTime[5];
     char *Dependency[DEPENDENCIES];
     int DCtr;
     char *Constructor;
 } Graph[DEPENDENCIES];

static char lTime[5] = {0,0,0,0,0};

static int GraphSize;

char *scan();

main (argc, argv)
 int argc;
 char **argv;
 {

     argc--; /* skip program name */
     argv++;

     if(argc > 0)
        Makefile = *argv;
     BuildGraph();
     resolve(0);
 }

BuildGraph()
 {
     register int i=0;
     char *s;

     if((f = fopen(Makefile, "r")) == NULL)
        {
           fprintf(stderr, "I don't know how to make %s.\nError %d\n",
                    Makefile, errno);
           exit(1);
        }
     if((s = scan()) == NULL)
        {
           fprintf(stderr, "The are no directions in %s\n", Makefile);
           exit(1);
        }
     for(GraphSize=0;i<DEPENDENCIES && (s != NULL); GraphSize++)
        {
           strncpy(Graph[GraphSize].ModTime,"",5);
           SetName(s, GraphSize);
           SetDep(GraphSize);
           SetCmd(GraphSize);
           s = scan();
        }
 }

resolve(n)
 int n;
 {
     char date[5];
     register i, x, flag;

     if(strncmp(Graph[n].ModTime, lTime,5) == 0)
        getdate(Graph[n].Name, Graph[n].ModTime);
     for (i=0, flag=FALSE; i< Graph[n].DCtr; i++)
        {
           if ((x = findName(Graph[n].Dependency[i])) >= 0)
              {
                 resolve(x);
                 if(!flag)
                    {
                       getdate(Graph[x].Name, Graph[x].ModTime);
                       if(strncmp(Graph[n].ModTime,Graph[x].ModTime,5) < 0)

                          flag++;
                    }
              }
           else if(!flag)
              {
                 getdate(Graph[n].Dependency[i], date);
                 if(strncmp(Graph[n].ModTime, date, 5) < 0)
                    flag++;
              }
        }
     if(flag)
        {
           if((x = system(Graph[n].Constructor)) != 0)
              {
                 fprintf(stderr, "Return %d from:\n%s\n", x,
                                      Graph[n].Constructor);
                 exit(x);
              }
        }
     return;
 }

findName(s)
 char *s;
 {
     register int i;
     char ts[127];

     strcpy(ts,s);
```

```
    fixname(ts);
    for(i=0;i<GraphSize;i++)
       if(cstrcmp(ts,Graph[i].Name) == 0)
          return(i);
    return(-1);
 }


SetName(s,i)
 char *s;
 int i;
 {
    Graph[i].Name = malloc(strlen(s) + 1);
    strcpy(Graph[i].Name, s);
    if(((s = scan()) == NULL) || (*s != ':'))
       {
          fprintf(stderr, "Colon required after %s\n", Graph[i].Name);
          exit(1);
       }
    return;
 }

SetDep(i)
 int i;
 {
    char *s;
    register int j=0;

    s = scan();
    Graph[i].DCtr = 0;
    while((s != NULL) && (*s != '=') && (j < DEPENDENCIES))
       {
          Graph[i].Dependency[j]  = malloc(strlen(s));
          strcpy(Graph[i].Dependency[j++], s);
          Graph[i].DCtr++;
          s = scan();
       }
    return;
 }

SetCmd(i)
 int i;
 {
    char s[81];

    if(fgets(s, 80, f) == NULL)
       {
          fprintf(stderr, "No command line for %s\n", Graph[i].Name);
          exit(1);
       }
    Graph[i].Constructor = malloc(strlen(s));
    strcpy(Graph[i].Constructor, s);
    return;
 }

static char token[81];

char *scan()
 {
    char *ptr, *limit;

    ptr = token;
    limit = ptr+80;             /*  High bound on token string */
    skipblanks();
    if((*ptr = getc(f)) == EOF)
       return NULL;

    /*  = and : are special tokens.  They are returned as single character

          tokens even if they have non-blanks on either side.
    */
    if((*ptr == '=') || (*ptr == ':'))
       ptr++;
    else                        /*  Accumulate characters until a delimiter
*/
       for(ptr++; (ptr != limit) && (!delim(*ptr = getc(f))); ptr++);
    *ptr = '\0';                /*  seal off the string */
    return token;
 }
/*  Delim returns TRUE if c is a delimiter character, FALSE
    otherwise.
*/
delim(c)
 char c;
 {
     switch (c)
       {
          case '\n':
          case ' ':
          case ',':
          case '\0':
          case '\t':
             return(TRUE);
             break;
          case '=':
          case ':':
             ungetc(c,f);
             return(TRUE);
             break;
          default:
             return(FALSE);
       } /* end case */
    return(FALSE);   /* Never executed */
 }

skipblanks()   /* Read from file 'f' until just before a non-
                  delimiter. */
 {
    register char c;

    while(isspace(c = getc(f)) || (c == ','));
    ungetc(c,f);
    return;
 }


getdate(filen, date)    /* put the last-modified date of filen in date */

 char *filen;             /* file name */
 char date[5];            /* last-modified date and time */
 {
    char *dir, *nm, *find_dir(), *find_nm();
    char diskname[33];                 /* Name of disk containing filen */
    FILE *DirFile, *DiskFile;
    struct dirent Entry;               /* Directory entry format */
    int found=FALSE;
    long LSN;                          /* Logical sector number */
    struct fildes Descriptor;          /* File descriptor format */

    find_disk(filen,diskname);         /* Extract disk name from
                                          qualified filename */
    nm = find_nm(filen);               /* Extract filename from
                                          qualified file name */
    fixname(nm);                       /* change nm to all caps, C-format stri
ng */
    dir = find_dir(filen);             /* Extract directory name from qualifie
d

                                          file name */

    if((DirFile = fopen(dir, "d")) == NULL)
       {
          fprintf(stderr, "Can't open directory %s\n", dir);
          exit(1);
       }

    /*
     Search through the directory for filename nm.
    */
    fixname(nm);
    while(fread(&Entry, sizeof Entry, 1, DirFile) == 1)
       {
          if(*Entry.dir_name != '\0')
             {
                if(cstrcmp(nm, Entry.dir_name) == 0)
                   {
                      found++;
                      break;
                   }
             }
       }
    if(!found)
       {
          fprintf(stderr, "%s can't be found in %s\n", nm, dir);
          exit(1);
       }

    /*
       Close the directory and open the disk.
    */

    fclose(DirFile);
    if((DiskFile = fopen(diskname, "r")) == NULL)
       {
          fprintf(stderr, "Error %d in open for %s\n", errno, diskname);
          exit(1);
       }

    l3tol(&LSN, Entry.dir_addr,1);
    fseek(DiskFile, LSN*256, 0);  /* Seek to file descriptor on disk */
    if(fread(&Descriptor, sizeof Descriptor, 1, DiskFile) == NULL)
       {
          fprintf(stderr, "Error %d in disk read for %s\n", ferror(DiskFil
e),
           diskname);
          exit(1);
       }

    fclose(DiskFile);
    _strass(date, Descriptor.fd_date, 5);
    return;
 }

fixname(s)   /* Switch s from OS-9 string to C-string and convert it to ca
ps */
 char *s;
 {
    register char *ptr;

    for(ptr = s; *ptr < 127 && *ptr > 0; ptr++)
       *ptr = toupper(*ptr & 127);
    *ptr = toupper(*ptr & 127);
    ptr++;
    *ptr = '\0';
    return;
 }

char *find_nm(s) /* must be called before find_dir */
 char *s;
 {
    register char *loc;

    if((loc = rindex(s,'/')) == NULL)
       return s;
    else
       return loc+1;
 }

char *find_dir(s)  /* return a directory name or "." if non can be
                          found in s. */
 char *s;
 {
    register char *loc;

    if((loc = rindex(s, '/')) == NULL)
       return ".";

    *loc = '\0';
    return s;
 }

find_disk(s,disk) /* determine the disk file s is on given its file name.
                         If no device name is in the file name, assume
                         the disk with the data directory on it.
                         Return the @-qualified file name for the device. */
 char *s, *disk;
 {
    register char *ptr;

    ptr = disk;
    if(*s == '/')
       do
          *ptr++ = *s++;
       while((*s != '/') && (*s != '\0'));

    *ptr++ = '@';
    *ptr = '\0';
    return;
 }


static char ts2[127];

cstrcmp(s1, s2) /* compage change s2 to C-format, all caps and compare to
s1 */
    char *s1, *s2;
    {
       strcpy(ts2, s2);
       fixname(ts2);
       return(strcmp(s1,ts2));
    }

```

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, GA 30207

INTRODUCTION

This month's column continues the definition of the string-handling library started in the previous column. It provides the text of many of the shorter functions, along with explanations of how they work.

STRING-HANDLING IN C

The "b" family of string-handling functions allows arbitrary contents of strings, as the processing is controlled entirely by the specified length. Note that the C compiler handles only null-terminated strings properly, in terms of string constant definition, string input, string output, etc. Thus length-terminated strings must be handled carefully, or their contents may be either prematurely terminated by a single 0x00 character or operations on them may not terminate properly because of the lack of a terminating 0x00 character.

bcmp(s1, s2, len) returns the number of bytes remaining in the strings after any equal bytes at the beginning of the strings have been skipped. The function counts down the length while it performs the comparisons.

```
int bcmp(s1, s2, len)
char *s1, *s2;
int len;
{
    while (--len >= 0 && *s1++ == *s2++);
    return len+1;
}
```

bcopy(src, dst, len) copies "len" bytes from the source "src" to the destination "dst".

```
bcopy(src, dst, len)
char *src, *dst;
int len;
{
    while (--len >= 0) *dst++ = *src++;
}
```

bfill(dst, len, fill) copies "len" fill characters to "dst".

```
bfill(dst, len, fill)
char *dst;
int len;
char fill;
{
    while (--len >= 0) *dst++ = fill;
}
```

bmove(dst, src, len) copies "len" bytes from the source "src" to the destination "dst". It differs from "bcopy" in the order of its source and destination arguments.

```
bmove(dst, src, len)
char *dst, *src;
int len;
{
    while (--len >= 0) *dst++ = *src++;
}
```

bzero(dst, len) copies "len" 0x00 bytes to "dst".

```
bzero(dst, len)
char *dst;
int len;
{
    while (--len >= 0) *dst++ = '\0';
}
```

The "mem" family of string-handling functions allows arbitrary contents of strings, as the processing is controlled entirely by the specified length. Although the functions in the family are similar to the functions in the "b" family, although they have different argument orders, return different values, perform slightly different operations, etc.

memccpy(dst, src, chr, len) copies bytes from "src" to "dst" until either "len" bytes have been moved or a byte equal to "chr" has been moved. It returns either NULL or a pointer one beyond the location in the destination string with "chr".

```
char *memccpy(dst, src, chr, len)
char *dst, *src, chr;
int len;
{
    while (--len >= 0)
        if ((*dst++ = *src++) == chr)
            return dst;
    return NULL;
}
```

memchr(src, chr, len) searches the memory area pointed to by "src" extending for "len" bytes, looking for an occurrence of the byte "chr". It starts at the beginning of the string and stops when it encounters the first match.

```
char *memchr(src, chr, len)
char *src, chr;
int len;
{
    while (--len >= 0)
        if (*src++ == chr)
            return src-1;
    return NULL;
}
```

memcmp(lhs, rhs, len) compares the two memory areas "lhs[0..len-1]" and "rhs[0..len-1]". It returns a value < 0, == 0, or > 0, depending upon whether "lhs" < "rhs", "lhs" == "rhs", or "lhs" > "rhs". It skips the equal prefixes and uses the values of the first unequal characters to determine the comparison value.

```
int memcmp(lhs, rhs, len)
char *lhs, *rhs;
int len;
{
    while (--len >= 0)
        if (*lhs++ != *rhs++)
            return lhs[-1]-rhs[-1];
    return 0;
}
```

memcpy(dst, src, len) copies "len" bytes from "src" to "dst" and returns a pointer to "dst".

```
char *memcpy(dst, src, len)
char *dst, *src;
int len;
{
    char *d = dst;
    while (--len >= 0) *dst++ = *src++;
    return d;
}
```

memmov(dst, src, len) copies "len" bytes from "src" to "dst" and returns "dst"+"len".

```
char *memmov(dst, src, len)
char *dst, *src;
int len;
{
    while (--len >= 0) *dst++ = *src++;
    return dst;
}
```

memrchr(src, chr, len) searches the memory area pointed to by "src" extending for "len" bytes, looking for the last occurrence of the byte "chr". It starts at the beginning of the string, but scans the entire string, rather than stopping with the first match.

```
char *memrchr(src, chr, len)
char *src, chr;
int len;
{
    char *ans;
    for (ans = NULL; --len >= 0; src++)
        if (*src == chr)
            ans = src;
    return ans;
}
```

memrev(dst, src, len) copies "len" bytes from "src" to "dst", in reverse order. It will work with completely overlapping, but not partially overlapping, source and destination strings. On each iteration, it swaps successive characters from the next positions from the front and end of each string.

```
memrev(dst, src, len)
char *dst, *src;
int len;
{
    char *dste = dst+len, *srce = src+len, t;
    while (srce > src)
    {
        t = *--srcz;
        *--dste = *src++;
        *dst++ = t;
    }
}
```

memset(dst, chr, len) fills the memory area "dst[0..len-1]" witb "len" bytes all equal to "chr", and returns a pointer to "dst".

```
char *memset(dst, chr, len)
char *dst, chr;
int len;
{
    char *d = dst;
    while (--len >= 0) *dst++ = chr;
    return d;
}
```

The "str" family of string-handling functions does not allow arbitrary contents of strings, as the processing is controlled by the terminating nulls in each string. This is consistent with the manner in which C compilers handle constant strings and the standard C functions handle character strings.

strcat(s, t) concatenates "t" on the end of "s" and returns a pointer to "s". First it finds the end of "s"; then it copies "t" to the end of "s".

```
char *strcat(s, t)
char *s, *t;
{
    char *save = s;
    while (*s++);
    for (--s; *s++ = *t++; );
    return save;
}
```

strchr(s, c) returns a pointer to the first place in "s" where "c" occurs, or NULL if "c" does not occur in "s".

```
char *strchr(s, c)
char *s, c;
{
    for (;;)
    {
        if (*s == c) return s;
        if (!*s++) return NULL;
    }
}
```

strcmp(s, t) returns a value > 0, = 0, or < 0 when "s" > "t", "s" = "t", or "s" < "t", according to the ASCII sequence of characters. It skips the equal prefixes and uses the values of the first unequal characters to determine the comparison value.

```
int strcmp(s, t)
char *s, *t;
{
    while (*s == *t++)
        if (!*s++) return 0;
    return *s-t[-1];
}
```

strcpy(dst, src) copies the characters starting with "src" to the area starting with "dst" until a null character is found, and returns a pointer to "dst".

```
char *strcpy(dst, src)
char *dst, *src;
{
    char *save = dst;
    while (*dst++ = *src++);
    return save;
}
```

strend(s) returns a character pointer to the null which ends "s".

```
char *strend(s)
char *s;
{
    while (*s++);
    return s-1;
}
```

strlen(s) returns the number of characters in "s".

```
int strlen(s)
char *s;
{
    int l = 0;
    while (*s++) ++l;
    return l;
}
```

strmov(dst, src) copies the null-delimited string pointed to by "src" into "dst", and returns a pointer to the terminating null in "dst".

```
char *strmov(dst, src)
char *dst, *src;
{
    while (*dst++ = *src++);
    return dst-1;
}
```

strrchr(s, c) returns a pointer to the last occurrence of "c" in "s", or NULL if "c" is not found in "s".

```
char *strrchr(s, c)
char *s, c;
{
    char *ans;
    for (ans = NULL; *src; src++)
        if (*src == chr)
            ans = src;
    return ans;
}
```

strrev(dst, src) copies characters from "src" to "dst", in reverse order. It will work with completely overlapping, but not partially overlapping, source and destination strings. On each iteration, it swaps successive characters from the next positions from the front and end of each string.

```
strrev(dst, src)
char *dst, *src;
{
    char *dstz, *srcz = src, t;
    while (*srcz++);
    srcz--;
    dstz = dst + (srcz - src);
    while (srcz > src)
    {
        t = *--srcz;
        *--dstz = *src++;
        *dst++ = t;
    }
}
```

strrpt(dst, src, k) repeats string "src" into "dst" "k" times. It returns the number of characters moved.

```
int strrpt (dst, src, k)
char *dst, *src;
int k;
{
    char *save = dst, *p;
    for ( ; --k >= 0; --dst)
        for (p = src; *dst++ = *p; );
    return dst-save;
}
```

strsub(dst, src, off, len) copies up to "len" bytes from "src"+"off" to "dst". The value returned is a pointer to the terminating null of the resulting string.

```
char *substr(dst, src, off, len)
char *dst, *src;
int off, len;
{

    while (--off >= 0)
        if (!*src++)
        {
            *dst = '\0'
            return dst;
        }
    while (--len >= 0)
        if (!(*dst++ = *src++))
            return dst-1;
    *dst = '\0';
    return dst;
}
```

Next month's column will continue the expansion of O'Keefe's string-processing functions. The ultimate goal is the definition of several families of functions which will provide the programmer with a flexible library which will increase productivity, readability, ease of use, and enhance the structuring of C programs.

C PROBLEM

The problem with the following definition:

```
#define tolower(x) (isupper(x) ? (x)|32 : (x))
```

is in its side-effects with certain arguments. Consider the effect of the following usage of the definition:

```
c = tolower(*p++);
```

which, when expanded, becomes the following:

```
c = (isupper(*p++) ? (*p++)|32 : (*p++));
```

Note that the character which is tested for case is not the same character which may be converted to lower case. There are at least two possible solutions to the problem. One is to make "islower" a function, taking advantage of the call-by-value of arguments to C functions, as follows:

```
char islower(x)
char x;
{
    return (isupper(x) ? (x)|32 : (x));
}
```

which works only for arguments of type char and int. Another possible solution involves the introduction of an intermediate variable to circumvent the double expansion of the argument of the definition. The revised definition and variable declaration are as follows:

```
#define tolower(x) (isupper(_c=(x)) ? _c|32 : _c)
char _c;
```

which works for arguments of any low-level type,. but is slightly less efficient in code and time than the original definition.

What does the following program output?:

```
#include "stdio.h"
#define sep(x) if ((x) == '\t') printf(" ")
main()
{
        char c[] = "abc\tdef";
        char *p;
        for (p = c; *p; p++)
        {
            if (*p != 'c')
                exp(*p);
            else
                printf("%c",*p);
        }
    }
```

What guideline for C programs does it illustrate?

EXAMPLE C PROGRAM

Following is this month's example C function; it is from Phil Gunsul, and provides a "rename" function for the Introl version of C for FLEX.

```
/*  Rename will rename a file on the disk.  The string s1 must
    point to the old file name and extention, with an optional
    drive number followed by a period.  For example, s1 may
    point at a string "2.junk.txt".  If a number is not
    specified, such as "junk.txt" the working drive number will
    be used.  S2 should point at the desired new name, such as
    "junk.bak".  No number is allowed to prefix s2 , and it must
    have an extention prefixed by a '.'.  If the file manager is
    unable to change names (disk is write protected, a file by
    that name already exist, etc.), a -1 will be returned.
*/


#include <stdio.h>
#include <flex.h>

rename(old_name_ext, new_name_ext)
char     *old_name_ext, *new_name_ext;
{


         char     c;
         struct fcb wrk_fcb;

         wrk_fcb.function = RENAME;
         if (isdigit(c = *old_name_ext)) {
                  wrk_fcb.drive = *old_name_ext++ - '0';
                  old_name_ext++;
         } else
                  wrk_fcb.drive = FLEX_DATA.work_drive;

         transfer(&wrk_fcb.filename, old_name_ext);
         transfer(&wrk_fcb.s.new_name, new_name_ext);

         return(_fms(&wrk_fcb, c));
}


transfer(fcb_pnt, string)
char     *fcb_pnt, *string;
{
         int      j, i;
         char     c;

         for (i = 10; i > 0; i--)
                  fcb_pnt[i] = 0;

         while ((c = string[i]) != '.' && i < 8)
                  fcb_pnt[i++] = c;

         j = ++i;
         i = 8;

         while ((c = string[j++]) != 0 && i < 11)
                  fcb_pnt[i++] = c;
}
```

68000 USER NOTES

Philip Lucido
2320 Saratoga Drive
Sharpville, PA 16150

Portability

In one of my previous articles, I mentioned that I was now commonly writing utility programs in C, and using them without change on both the 6809 and the 68000. This has proved to be more and more important. For instance, I am working on several different large programs at the moment, with an eye towards selling them, and it makes sense to be able to sell them in both 6809 and 68000 markets with minimal changes to the programs. As I write these programs, though, it is becoming obvious that simply using a high level language is no guarantee of portability. Writing a truly portable program turns out to require a little care and thought, as well as some good programming habits.

Syntax

There are some obstacles to program portability which are beyond a programmer's ability to control. The compilers used to implement a given high level language in two different environments may actually accept two slightly different languages, syntactically. For instance, with Pascal, there is no universally established method for declaring a default action whenever the expression in a case statement fails to match any of the case values. Since it is very useful to be able to specify a default, as in the C language **switch/case/default** construct, individual compiler authors have extended Pascal, each using their own peculiar syntax.

C also has some problems in this regard. While C compilers tend not to implement unique extensions to the language, probably because the standard language as defined in K & R (Kernighan & Ritchie, The C programming Language) is quite powerful and complete, various compilers fail to implement some features. This is commonly found in the so called 'Small C' compilers. Typically, these compilers do not accept C language features like floating point operations, structures, or initializers. Further, even few 'full' C compilers implement bit fields, which are described in K & R, or such newer features as passing of structures as parameters, added to C since the publishing of K & R.

Point one in writing portable programs, then, is to use the minimal language syntax which can be expected to be widespread among compilers. For C, this mostly means staying away from bit fields. Unless a Small C compiler is all you have available, go ahead and use structures, initializers, and the like, since full C compilers are now quite common, and structures, in particular, are indispensible in combatting other portability problems.

The Library

Much of the power of C is derived from the routines which make up the subroutine library which is supplied with the compiler. Unfortunately, different compilers come with different libraries. There does exist a standard of sorts, the library found with the Unix version of C. This became a standard mainly through it's inclusion in K & R, and is known the 'standard I/O', or stdio package. Just because a compiler claims to be Unix-compatible, or include the stdio package, though, is no promise of immediate portability as far as the library goes.

Generally, the presence of the stdio package means that certain file routines, known as the buffered I/O subroutines, are available. These include such functions as **fopen**, **fread**, **getch**, **putch**, and **printf**, which work by buffering data into blocks, which are then read or written as a whole. The buffered I/O routines are usually externally identical, so that they can safely be used in portable programs.

There is another set of I/O routines which supply more direct calls on the operating system. These routines include **read**, **write**, **open**, and **creat**. While direct I/O routines may in fact have these names in a C package, they may not be used in the same manner as the like routines in another package. For instance, **open** takes a parameter giving the file's access mode, such as read, write, or update. The actual numeric values of the code, though, may depend on the particular operating system. Thus, the access mode parameter used in the Microware C package is different from the same parameter in Unix C. While direct I/O may be required or preferred, for reasons of efficiency, portability may force a programmer to use the buffered routines instead. As an alternative, there are techniques using the C pre-processor which may assist in making direct I/O calls portable. These will be discussed later.

One constant source of headaches in writing portable C programs is the memory allocation routines. These routines are used to request more memory from the operating system, or to return memory which is no longer required. Generally, such routines as calloc, malloc, free, and brk are available. Depending on the c ller, there may be several variations of brk, such as Microware's sbrk, ibrk, and ebrk (in the OS-9/68000 version only). As with file I/O, these routines may be thought of as buffered and direct routines. The buffered routines, calloc, malloc, and free, request memory from the operating system in chunks, and then parcel it out in pieces as higher level requests are made. The brk routine, on the other hand, performs direct calls to the operating system. As such, it is more likely to change from compiler to compiler. Certainly check beforehand, but if it all possible, use the buffered routines for portability, as they are more likely to remain compatible among separate compilers.

Point two in writing portable programs: stick to whatever standard exists, as far as the run-time library of support routines goes. This standard, usually, will be the buffered routines found in the Unix C library. Use carefully, or preferably avoid, the direct I/O routines which exist with the same names but different forms in various compilers. Finally, avoid if at all possible those routines which are unique to a certain operating system or compiler, unless you do not plan on porting a program to another OS. For instance, OS-9 C compilers generally have an os9 subroutine, which is used to issue direct requests to the kernel. The subroutine may have a different name, though, and if you wish to port a program to CP/M, for example, where a subroutine called bios exists for the same purpose, the calls will be totally incompatible.

The Pre-processor

Sometimes, a program may be portable to another operating system with only some changes in various parameters such as buffer sizes. Here is where the pre-processor comes in handy. The C pre-processor does not actually understand the C language. Instead, its job is to read the C source text, searching for special lines hich are commands to the pre-processor, and modifying the text according to these commands before passing it on to the actual compiler. By proper use of pre-processor commands, different versions of a program can be selected by the modification of a single line in the source.

The main idea, as far as taking care of numbers and strings which may change between compilers and operating systems, is to create what are known as **manifest constants**. A manifest constant is a constant, or fixed value, which is given a name. In each place where the constant is required, the name is used instead. If the line equating the name with the constant is placed where it is highly visible, at the start of the source file, then modification of the program does not require digging into the text for obscure references. Instead, just the definitions are changed. As an example, suppose a program requires a buffer of a fixed size of 8K. Instead of numerous incomprehensible references to the number 8192, a single line

```
#define BUFFER_SIZE     8192
```

can be present at the start of the program, with the name BUFFER SIZE being used in the body of the program. Changing the size of the buffer, for a new operating system, for instance, is as simple as changing the single #define.

Still more can be done. It is rather inconvenient to have to physically change many definition lines in order to use a program on a new operating system. An alternative is to supply all of the constants, for each of the various operating systems. This is done with conditional compilation, using the #if/#else/#endif pre-processor commands. As an example, consider the BUFFER_SIZE definition:

```
#ifdef OS9
#define BUFFER_SIZE     8192
#else
#ifdef  UNIX
#define BUFFER_SIZE     32768
#else
#define BUFFER_SIZE     4096
#endif
#endif
```

The #ifdef pre-processor command checks if a given manifest constant has been defined before. A previous definition will usually be done using an #define statement, though some compilers allow a name to be defined from the command line. The lines above presuppose the definition of a manifest constant giving the name of the operating system being used. Thus, if OS-9 is used, a line such as '#define OS9 1' must exist. If running under Unix, the name UNIX is defined instead. If the constant OS9 has been defined, then the name BUFFER SIZE will be equated with the number 8192. If not, then the name UNIX is checked. If it exists, BUFFER SIZE will be 32768. If both OS9 and UNIX are undefined, a default definition of 4096 will be triggered.

The Microware C compilers automatically define names for the particular operating system. The 6809 C compiler predefines a name of **OS9**, while the 68000 version predefines a name of OSK. Thus, when working on a dual 6809/68000 machine, as I am, these names may be used in a program without definition, greatly enhancing the ability to compile a program under the two operating systems with no changes whatsoever to the source text.

The conditional pre-processor commands do not have to surround other pre-processor commands only. Normal C language may also be conditionally selected. For instance, in a program I have written, I need to use chain, which transfers execution to another program. There was a bug in the 68000 version of chain, though, and I was forced to drop back on the equivalent routines os9fork followed by wait. This is inefficient, though, and since the 6809 is tight on address space, I preferred to stick with the chain if possible. The resulting code went something like this:

```
#ifdef OSK
        os9fork(name,psize,...);
        wait(&status);
#else           /* OS-9 default */
        chain(name,psize,...);
#endif
```

If there are a large number of definitions that change between versions of a program running under different operating systems, it may be easier to prepare a separate text file, consisting only of the definition lines for a particular operating system, and use the #include pre-processor command to read the separate file. For instance, #include "defs.h" will cause the lines in the file defs.h to be included as part of a C program. If the source for a number of operating systems is kept as a single file on a machine, such as on a dual 6809/68000, #ifdef statements may be used as above to include different files based on the operating system. If the source is transported to a different machine, then the same definitions file name can be used, assuming the file name meets the requirements of the new operating system. If not, then the #include line will have to be changed.

Point three when writing portable C programs: use the pre-processor, especially as regards manifest constants. Don't tell yourself that of course some number will stay the same between operating systems. If there is any chance it might change, give it a name, and comment the definition of the name so you know to change it later.

Data Types

As long as programs were being ported among 8 bit computers, little attention needed to be paid to such matters as the size of ints and other data types. When moving bet een 8 and 16 bit systems, though, the matter can become quite important, causing bugs which are very difficult to find.

On the 6809, ints, short integers, and pointers are 16 bits wide, chars are 8 bits, and long integers are 32 bits. On the 68000, char, short, and long integers remain the same size, but types int and pointer are now 32 bits wide (at least under Microware C). Because of this, code which assumes 2 bytes per integer or pointer will fail. While a program will occasionally need to know the size of an int, it should never use a fixed constant. Instead, the **sizeof** operator should be used. This returns the size of a data type in units of the size of chars, which is 1 byte for practically all microcomputers. Thus, to increment a pointer of type char * past an integer, the line used should be

```
p += sizeof(int);
```

not

```
p += 2;
```

There are other places to look out for the data size problem. A program I once wrote created a temporary file which included pointers to symbol table references in memory. The program assumed a 2 byte pointer, and used calls to a routine outword() to perform the output to the file and inword() to read the pointer back. When the program was moved to a 16 bit processor, I had to slowly search through the entire program, looking for all inword()/outword() references to pointers. Each of these then had to be changed to 4 byte read/writes, using new routines inptr()/outptr(). It would have been far better if I had had the foresight to use separate routines for the pointers from the start. The moral: don't ever assume anything about the size of a data type. It might change.

With ints and longs being the same size in the 68000, it is tempting to ignore longs altogether. This can cause problems if you try to port back to the 6809. If a variable will fit in an integer in both the 6809 and the 68000, use an integer. If it needs a long with the 6809, use a long for both processors. Be especially careful when using **printf**, since it requires long parameters to be explicitly declared long in the output format string (e.g. "%8ld" instead of "%8d"). If you are forced to convert an int to a long s a 68000 program can be moved to the 6809, look carefully for these printfs.

Another problem with differences in data types has to do with sign extension and the type **char**. K & R specifically state that a character value may be converted to either a signed integer or an unsigned integer, and no assumptions should be made as to which actually occurs. The Microware C compilers, for both the 6809 and 68000, perform sign extensions, for instance, while CP/M C compilers I have used perform unsigned extension. To prevent problems, programs should perform an AND operation (c & 0xff) whenever sign extension might be a proble . Having said that, though, I have to acknowledge that it can be devilishly difficult to find all such occurrences.

More importantly for 6809/68000 portability, the same problem appears when dealing with type **short**. One program of mine created a symbol table which I wanted to keep as small as possible, so short integers were used where possible. This caused no problem on the 6809, since short and int are the same thing there. n the 68000, though, shorts are sign extended to ints. The results were quite confusing, until I printed out some values using debug printfs, only to discover that some values were being printed, in hex, as 0xffffabcd, instead of the expected 0xabcd. The answer, as before, is careful use of AND statements (val & xffff).

Sign extension can also be handled by specifically declaring affected variables as **unsigned** char or **unsigned short**. This will only work if your compiler understands such declarations, which is by no means assured, since K & R seems to specify that such declarations are illegal (I think - the book is rather hard to read, there). Despite K & R, several compilers, including the latest versions of the 68000 Microware compiler, do accept these types, probably because the Unix compiler does so.

The final point (this month) in working towards portability: watch the assumptions ab ut data types. An integer is not necessarily the same everywhere you look.

Where Did My Space Go?

I seem to have gotten too talkative again. There is still more to be covered, particularly the use of #defines to take advantage of features available only with particular microprocessors or operating systems. In addition, I am beginning to see information appear on the new 68020, and should have something to say about it next month.

- - -

SOFTWARE TOOLS IN PASCAL

SOFTWARE TOOLS IN PASCAL

Brian W. Kernigan and P.J.Plauger have written an excellent book that teaches good programming techniques. The programs and algorithims that they present in the book are useful, and they work. Many authors in the past using other programming languages have typically presented programs that are incomplete and barely work at best. This author was very pleasantly surprised at the quality of the programs that Kernigan and Plauger have presented in their book. This book is a treasure trove of useful and valuable Pascal programs.

Chapter one, titled "Getting Started", deals with the methods and styles that Kernigan and Plauger use throughout the rest of the book. Within this chapter the various lower level "primitives" are discussed in detail. These functions and procedures are used as the basic tools or building blocks for the rest of the book. Besides basic file copying methods, various character and word counting programs are illustrated.

Chapter two continues with various groups of programs that are called FILTERS. Filters are used to make changes to data that is being passed through them. Programs that fall into this category include TAB removal and replacement within text files. ther programs are designed to replace backspaces or perform text compression or decompression on files.

Chapter three discusses in detail various methods for handling data within files. This chapter includes a file comparison program. Also since some Pascal compilers do not include a "5nclude" function, a program is presented that performs this task. Other programs include file concatenation, dynamic file creation, and archiving.

Sorting is discussed in chapter four. Several methods are explained including the bubble sort, shell sort and quick sort algorithims. All three techniques are shown along with programs that demonstrate the algorithims. Besides in-memory sorting programs the chapter also illustrates the methods by which one can develop a variable record length sorting program that can sort files larger than those that can be sorted in memory.

Chapter five discusses the methods of text pattern checking and matching. A program named FIND illustrates the methods by which one can implement this useful function. Another program is explained which also changes text besides just finding it.

A very interesting and novel text editor is presented in chapter six. This particular editor is a line editor, and it is very well documented. Thus one should not have too much difficulty implementing it on any specific computer system. The editor sports all the neccesary functions including line insertion, deletion, and search and replace procedures.

Chapter seven discusses text formatting. The text formatter that is presented contains all the needed functions that one would normally require in formatting a text file. Some of the functions include left and right margin justification, paging, page numbering, line centering, and indenting. The text formatter works very well as this article was originally formatted using it.

Chapter eight covers macro processing. Macros are used to extend a programming language such as assembly language. Macros can also be used to expand upon the text editor that is presented in the book. Also one could use macros to replace text in a file with more complex forms of text.

In the appendix, the last section of the book, various example procedures and functions are provided to help aid one in being able to utilize these programs. Such low level procedures and functions include opening and closing files, and the reading and writing of characters. Some implementations include examples for the University of California at Berkeley (UCB), Whitesmiths Limited, University of California at San Diego (UCSD) Pascal systems. These various implementations also include UNIX compatibility, so users with microcomputer systems using TSC's UNIFLEX or Microware's OS-9 should have little difficulty getting the programs up and running.

"Software Tools in Pascal" was preceded by an earlier work titled "Software Tools". The earlier book presented it's programs in RATFOR, which is a language based on FORTRAN. The newer book goes much farther in redesigning the programs and considerably improving upon the original FORTRAN implementations. "Software Tools in Pascal" is published by the Addison-Wesley Publishing Company.

In closing, the author has added the following Pascal programs named DPAGE, NDPAGE, TPAGE, and NTPAGE to those in the book. DPAGE and NDPAGE perform the function of setting up formatted text into double column pages. TPAGE and NTPAGE set up the formatted text into triple column pages. All four Pascal programs were compiled using Microware's Pascal compiler and language package. All of these programs as presented in the book and the additional ones can be compiled all the way into Native Code using Microware's Pascal compiler. NDPAGE and NTPAGE also demonstrate the use of entering information into a program via the command line with the language extension SYSPARAM.

Earl W. Bollinger
912 West First Street, Apt 5
Fort Worth, Texas 76102
817-877-0625

```
WHILE comm[1]='Y' DO
  BEGIN
    random(rnd,x,rn);
    f:=150+rnd*10;
    random(rnd,x,rn);
    v:=rnd*10-100;
    random(rnd,x,rn);
    d:=400+rnd*10;
    random(rnd,x,rn);
    g:=1+rnd;
    IF g>9 THEN g:=9;
    landed:=false;
```

```
WHILE not landed DO
 BEGIN
   movelem(d);
   status(f,v,d);
   IF f>0 THEN
     burnrate(b)
   ELSE
     b:=0;
   IF b>f THEN
     b:=f;

   f:=f-b;
   c:=b-g;
   d:=trunc(d+v+c/2);
   v:=v+c;

   IF d<=0 THEN
     landed:=true;
 END; {of inner while loop}

writeln;
writeln;
writeln('LEM is on the surface of the moon!');
writeln;

IF v<-5 then
 BEGIN
   writeln('Excessive speed on landing!!');
   writeln;
   crashed(rnd,x,v,f,d,rn)
 END
ELSE
 AOK(v,f);

writeln;
write(' Another game? (Y/N): '); prompt;
i:=1;
WHILE (not eoln) and (i<6) DO
  BEGIN
    read(comm[i]);
    i:=succ(i);
  END;
writeln;
IF comm[1]='y' THEN comm[1]:='Y';
END; { of outer while loop }
END.
```

```
        writeln(f:6,' units of fuel remaining.');
        random(rnd,x,rn);
        d:=f*rnd+3;
        writeln('Produced an explosion covering ',d:0,' sq miles');
        writeln(' of lunar surface!!');
        writeln;
      END;
   writeln('L E M   D E S T R O Y E D ! ! !');
   writeln;
   writeln('****** YOU BLEW IT! ******');
   writeln;
   writeln;
 END;

 PROCEDURE AOK(VAR v,f: INTEGER);
 BEGIN
   writeln;
   writeln(' C O N G R A T U L A T I O N S ! ! !');
   writeln;
   writeln('A perfect landing!!');
   writeln;
   writeln('Touchdown velocity: ',v:0);
   writeln('Fuel remaining: ',f:0);
   writeln;
 END;

 PROCEDURE MOVELEM(d: INTEGER);
 { used to make the LEM appear to move down the screen }
 VAR
   s: INTEGER;
 BEGIN
   s:=trunc(12-d/40);
   WHILE s>0 DO
```

```
    BEGIN
      writeln;
      s:=pred(s);
    END;
END;

{ The main program }

BEGIN
 z:=128;

 random(rnd,z,rn);
 i:=rnd+13;
 WHILE i>0 DO
  BEGIN
    random(rnd,z,rn);
    i:=pred(i);
  END;

writeln;
writeln;
writeln('      LUNAR LANDER GAME SIMULATION');
writeln;
writeln;
writeln('     Try to land the LEM on the surface of the moon by entering');
writeln(' the fuel burn rates when requested.');
writeln;
writeln('     GOOD LUCK!');
writeln;
writeln;
come[1]:='Y';



PROGRAM LUNARLANDER;
{ LunarLander is another implementation of the classical game }
{ by which you have to land a space vehicle on the moon.      }
{ This particular version could not have been implemented at  }
{ all if it wasn't for T.F.Elbert and his series of articles  }
{ in "68 Micro Journal" issues Nov.81, Dec.81, Jan.82, Feb.82 }
{ Titled 'Simulation, Games, and Random Variables.            }
{                                                             }
{ By E.W.Bollinger on May, 15 1982.                           }



VAR
  b,c,d,f,g,i,rnd,v,z: INTEGER;
  rn: REAL;
  come: ARRAY[1..5] of CHAR;
  landed: BOOLEAN;

PROCEDURE RANDOM(VAR rnd,z: INTEGER; rn: REAL);
VAR
  j: INTEGER;
{ used to generate random integers between 0 and 9 }

BEGIN
  mathabort(false);
  z:=101*z;
  IF z<0 THEN z:=z+32767+1;
  rn:=z/32767.0;
  mathabort(true);
  j:=mathresult; {used to clear any error codes out}

  rnd:=trunc(z/1000);
  IF rnd>9 THEN rnd:=rnd mod 10;
END;

PROCEDURE STATUS(VAR f,v,d: INTEGER);
BEGIN
  writeln(' 0          FUEL: ',f:0);
  writeln('(0)         SPEED: ',v:0);
  writeln('/-\         HEIGHT: ',d:0);
  writeln;
END;

PROCEDURE BURNRATE(VAR b: INTEGER);
BEGIN
  write('BURN: '); prompt;
  readln(b);
END;
```

```
PROCEDURE CRASHED(VAR rnd,z,v,f,d: INTEGER; rn: REAL);
BEGIN
  writeln;
  writeln('CRASH     CRASH     CRASH');
  writeln('*****     *****     *****');
  writeln;
  writeln;
  writeln('Impact velocity: ',v:0);
  writeln('LEM buried: ',d:0,' feet.');
  writeln;
  IF f>0 THEN
    BEGIN
```

```
     00  0 PROGRAM NTPAGE;
     00  0 { NTPAGE-- outputs a three column format page of text }
     00  0 { from the file as inputted. The file should be set   }
     00  0 { up with columns less than 40 wide and 66 lines to a }
     00  0 { page.                                               }
     00  0 { This version prints page numbers, starting with the }
     00  0 { page number as entered by the user. It also prints  }
     00  0 { only the leftmost top of page header line, and      }
     00  0 { strips all the rest of the header and footer lines. }
     00  0 { NTPAGE is inspired be Kernigan and Plauger's book   }
     00  0 { titled 'Software Tools in Pascal'.                  }
     00  0 {                                                     }
     00  0 { Typical command line:                               }
     00  0 {    DS9: Pascals <source >destination NTPAGE :pagenum }
     00  0 {    DS9: Pascals <source >destination NTPAGE s1bb    }
     00  0 {       If compiled into object code:                 }
     00  0 {    DS9: NTPAGE <source >destination :page_number    }
     00  0 {                                                     }
     00  0 { BY E.W.BOLLINGER on October 12, 1982                }
     00  0
     00  0
     00  0 CONST
     00  0   PAGELEN = 66;
     00  0   MAXLINE = 41;
     00  0   MAXSTR  = 132;
     00  0
     00  0 TYPE
     00  0   string = array[1..MAXSTR] of char;
     00  0   page = array[1..PAGELEN,1..MAXLINE] of char;
     00  0
     00  0 VAR
     00  0   NEWLINE,NULL : char;
    -20  0   lpage,mpage,rpage : page;
 -01200  0   pn : integer;
 -01220  0   done : boolean;
 -01230  0
 -01230  0 FUNCTION GETC(var c: char): boolean;
     00  1 { GETC -- gets a character from standard input }
     00  1 Begin
      0  2   if eof then
      8  3      c:=NULL
      9  3   else if eoln then
     22  4     begin
     22  4      readln;
     25  4      c:=NEWLINE
     26  4     end
     30  4   else
     33  4     read(c);
     37  2   if c=NULL then
     48  3      getc:=true
     48  3   else
     55  3      getc:=false;
     59  2 end; { of getc }
      0  1
      0  1 FUNCTION GETLINE(var s: string; maxsize: integer): boolean;
      0  1 { getline -- gets a line of text from the standard input }
      0  1 var
     00  1   i: integer;
    -20  1   ch: char;
    -30  1 Begin
      0  2   i:=1;
      4  2   while (not(getc(ch))) and (i<maxsize) and (ch<>NEWLINE) do
     25  3     begin
     25  3       s[i]:=ch;
     37  3       i:=succ(i)
     39  3     end;
     43  2   if (ch=NULL) and (i>1) then  { back up one, gone too far }
 68    56   3       i:=pred(i);
 69    59   2    s[i]:=NEWLINE;
 70    72   2    getline:=(ch<>NULL);
 71    81   2 End; { of getline }
 72     0   1
 73     0   1 PROCEDURE GETPAGENUMBER(var pn: integer);
 74     0   1 { getpagenumber -- gets the starting page number from the }
 75     0   1 { parameter passed to it in the command line.              }
 76     0   1 VAR
 77    0B   1    i: integer;
 78   -2B   1    RETURN: char;
 79   -3B   1 Begin
 80     0   2    RETURN:=NEWLINE;
 81     7   2    i:=0;
 82     9   2    WHILE (sysparam[i]<>' ') and (i<79) DO
 83    33   3          i:=succ(i);
 84    39   2    sysparam[i]:=RETURN;
 85    52   2    i:=1;
 86    54   2    IF sysparam[0]<>RETURN THEN
 87    68   3       i:=trunc(cnvtreal(sysparam));
 88    78   2    pn:=i;
 89    81   2 End; { of getpagenumber }
 90     0   1
 91     0   1
 92     0   1 PROCEDURE GETPAGE(var pg:page; var done: boolean);
 93     0   1 { getpage -- gets an entire page of text from standard input }
 94     0   1 var
 95    0B   1    i,n: integer;
 96   -4B   1    s: string;
 97 -136B   1 Begin
 98     0   2    For i:=1 to PAGELEN Do
 99    17   3      Begin
100    17   3        IF (not(done)) and (getline(s,MAXLINE)) Then
101    35   4          begin
102    35   4            n:=1;
103    37   4            IF (s[n]<>NEWLINE) and (s[n]<>NULL) Then
104    77   5              begin
105    77   5                REPEAT
106    77   5                  pg[i,n]:=s[n];
107   111   6                  n:=succ(n)
108   113   6                UNTIL (s[n]=NEWLINE) or (s[n]=NULL) or (n=MAXLINE);
109   159   5              end;
110   159   4            IF n<MAXLINE Then
111   166   5              pg[i,n]:=NEWLINE;
112   189   4          end
113   189   4        ELSE
114   192   4          Begin

115   192   4            pg[i,1]:=NEWLINE;
116   210   4            pg[i,2]:=NULL;
117   228   4            done:=true
118   229   4          End;
119   232   3      End; { of for next loop }
120   246   2  End; { of getpage }
121     0   1
122     0   1 PROCEDURE OUTPUTTRIPLEPAGE;
123     0   1 { It simply takes three pages of previously formatted text }
124     0   1 { and outputs them all onto one page.                      }
125     0   1 var
126    0B   1    i,n,x,mid: integer;
127   -8B   1 Begin
128     0   2    mid:=trunc(MAXSTR/2-4);
129    14   2    i:=1;
130    16   2    WHILE i<=PAGELEN DO
131    23   3      Begin
132    23   3       IF i=64 THEN
133    30   4        Begin
136    30   4          write('  ');
135    38   4          FOR x:=1 to mid DO
136    48   5              write(' ');
137    66   4          write('PAGE ',pn:4);
138    81   4          pn:=succ(pn)
139    85   4        End
140    88   4       ELSE
141    91   4        Begin
142    91   4          IF (lpage[i,1]=NEWLINE) and
143   114   4             (mpage[i,1]=NEWLINE) and
144   138   4             (rpage[i,1]=NEWLINE) THEN
145   165   5           begin
146   165   5              {do nothing for this part}
147   165   5           end
148   165   5         ELSE
149   168   5          Begin
150   168   5            write('  ');
151   176   5            n:=1;
152   178   5            WHILE (lpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
153   214   6              Begin
154   214   6                write(lpage[i,n]);
155   242   6                n:=succ(n)
156   244   6              end;
157   248   5            IF n<MAXLINE Then
158   255   6               begin
159   255   6                 FOR x:=n to MAXLINE DO
160   267   7                   write(' ');
161   285   6                 end;
162   285   5

163   285   5             IF i<>3 THEN   (strip extra two headers)
164   291   6              Begin
165   291   6                 write('    ');
166   299   6

167   299   6                 n:=1;
168   301   6                 WHILE (mpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
169   337   7                   begin
170   337   7                     write(mpage[i,n]);
171   365   7                     n:=succ(n)

172   367   7                   end;
173   371   6                  IF n<MAXLINE then
174   378   7                     begin
175   378   7                       FOR x:=n to MAXLINE DO
176   389   8                           write(' ');
177   407   7                     end;
178   407   6

179   407   6                  write('    ');

180   415   6
181   415   6                  n:=1;
182   417   6                  WHILE (rpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
183   453   7                    Begin
184   453   7                      write(rpage[i,n]);
185   481   7                      n:=succ(n)
186   483   7                    end;
187   487   6                End;
188   487   5            End;
189   487   4         End;
190   487   3        writeln;
191   490   3        i:=succ(i);
192   493   3      End;
193   496   2  End; { of outputtriplepage }
194     0   1
195     0   1 { MAIN PROGRAM }
196     0   1  Begin
197     0   1    NEWLINE:=chr(13);
198     7   1    NULL:=chr(0);
199    12   1    done:=false;
200    17   1    pn:=1;
201    21   1
202    21   1    getpagenumber(pn);
203    27   1
204    27   1    WHILE not(done) DO
205    34   2     begin
206    34   2       getpage(lpage,done);
207    43   2       getpage(mpage,done);
208    52   2       getpage(rpage,done);
209    61   2       outputtriplepage;
210    64   2     end;
211    67   1  End. { of tpage }

PROC NAME      PSEC  PSIZE  LOCAL  STACK   CSEC  CSIZE  DEBUG
   0 MTPAGE       0     69   8123      9      9      0      0
   1 GETC         1     60      0     15      2      0      0
   2 GETLINE      2     83      3     10      3      0      0
   3 GETPAGEN     3     82      3     16      4      0      0
   4 GETPAGE      4    248    138     13      5      0      0
   5 OUTPUTTR     5    497     10     17      7     20      0
                      1039   8277     80            20

211 Lines of source code compiled with no errors found

  1      0B  0 PROGRAM TPAGE;
  2      0B  0 { TPAGE -- outputs a three column format page of text }
  3      0B  0 { from the file as inputted. The file should be set   }
  4      0B  0 { up with columns less than 40 wide and 66 lines to a }
```

```
    00  0 { page.                                                  }
    00  0 { TPAGE was inspired by Kernigan and Plauger's book      }
    00  0 { titled 'Software Tools in Pascal'.                     }
    00  0 {                                                        }
    00  0 { Typical command input line:                            }
    00  0 {  Pascals <source >destination Tpage                    }
    00  0 { or if compiled into object code:                       }
    00  0 {  Tpage <source >destination                            }
    00  0 {                                                        }
    00  0 { BY E.W.BOLLINGER on October 1, 1982                    }
    00  0
    00  0
    00  0 CONST
    00  0   PAGELEN = 66;
    00  0   MAXLINE = 41;
    00  0   MAXSTR  = 132;
    00  0
    00  0 TYPE
    00  0   string = array[1..MAXSTR] of char;
    00  0   page = array[1..PAGELEN,1..MAXLINE] of char;
    00  0
    00  0 VAR
    00  0   NEWLINE,NULL : char;
   -2D  0   lpage,mpage,rpage : page;
-8120D  0   done : boolean;
-8121D  0
-8121D  0 FUNCTION GETC(var c: char): boolean;
    0D  1 { GETC -- gets a character from standard input }
    0D  1 Begin
     0  2   if eof then
     8  3     c:=NULL
     9  3   else if eoln then
    22  4     begin
    22  4       readln;
    25  4       c:=NEWLINE
    26  4     end
    30  4   else
    33  4     read(c);
    37  2   if c=NULL then
    48  3     getc:=true
    48  3   else
    55  3     getc:=false;
    59  2 end; { of getc }
     0  1
     0  1 FUNCTION GETLINE(var s: string; maxsize: integer): boolean;
     0  1 { getline -- gets a line of text from the standard input }
     0  1 var
    0D  1   i: integer;
   -2D  1   ch: char;
   -3D  1 Begin
     0  2   i:=1;
     4  2   while (not(getc(ch))) and (i<maxsize) and (ch<>NEWLINE) do
    25  3     begin

    25  3       s[i]:=ch;
    37  3       i:=succ(i)
    39  3     end;
    43  2   if (ch=NULL) and (i>1) then  { back up one, gone too far }
    56  3     i:=pred(i);
    59  2   s[i]:=NEWLINE;
    72  2   getline:=(ch<>NULL);
    81  2 End; { of getline }
     0  1
     0  1 PROCEDURE GETPAGE(var pg:page; var done: boolean);
     0  1 { getpage -- gets an entire page of text from standard input }
     0  1 var
    0D  1   i,n: integer;
   -4D  1   s: string;
-13AD  1 Begin
     0  2   For i:=1 to PAGELEN Do
    17  3     Begin
    17  3       IF (not(done)) and (getline(s,MAXLINE)) Then
    35  4         Begin
    35  4           n:=1;
    37  4           IF (s[n]<>NEWLINE) and (s[n]<>NULL) Then
    77  5             Begin
    77  5               REPEAT
    77  5                 pg[i,n]:=s[n];
   111  6                 n:=succ(n)
   113  6               UNTIL (s[n]=NEWLINE) or (s[n]=NULL) or (n=MAXLINE);
   159  5             end;
   159  4           IF n<MAXLINE Then
   166  5             pg[i,n]:=NEWLINE;
   189  4         end
   189  4       ELSE
   192  4         Begin
   192  4           pg[i,1]:=NEWLINE;
   210  4           pg[i,2]:=NULL;
   228  4           done:=true
   229  4         End;
   232  3     End; { of for next loop }
   246  2 End: { of getpage }
     0  1
     0  1 PROCEDURE OUTPUTTRIPLEPAGE;
     0  1 { It simply takes three pages of previously formatted text }
     0  1 { and outputs them all onto one page.                      }
     0  1 var
    0D  1   i,n,x: integer;
   -6D  1 Begin
     0  2   i:=1;
     4  2   WHILE i<=PAGELEN DO
    11  3     begin
    11  3       IF (lpage[i,1]=NEWLINE) and (mpage[i,1]=NEWLINE) and (rpage[i,1]
 =NEWLINE) THEN
    85  4         {do nothing for this part}
    85  4       ELSE
    88  4        Begin
    88  4          write('  ');
    96  4          n:=1;
    98  4          WHILE (lpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
   134  5            begin
   134  5              write(lpage[i,n]);

   162  5              n:=succ(n)
   164  5            end;
   168  4          IF n<MAXLINE Then
   175  5            begin
   175  5              FOR x:=n to MAXLINE DO
   186  6                write(' ');
   204  5              end;
   204  4
   204  4           write('    ');
   212  4
   212  4           n:=1;
   214  4           WHILE (mpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
   250  5             begin
   250  5               write(mpage[i,n]);
   280  5               n:=succ(n)
   282  5             end;
   286  4            IF n<MAXLINE then
   293  5              begin
   293  5                FOR x:=n to MAXLINE DO
   304  6                    write(' ');
   322  5              end;
   322  4
   322  4            write('    ');
   330  4
   330  4            n:=1;
   332  4            WHILE (rpage[i,n]<>NEWLINE) and (n<MAXLINE) DO
   368  5              begin
   368  5                write(rpage[i,n]);
   396  5                n:=succ(n)
   398  5              end;
   402  4          End;
   402  3
   402  3         writeln;
   405  3         i:=succ(i);
   408  3       End;
   411  2 End; { of outputtriplepage }
     0  1
     0  1 { MAIN PROGRAM }
     0  1  Begin
     0  1    NEWLINE:=chr(13);
     7  1    NULL:=chr(0);
    12  1    done:=false;
    17  1
    17  1    WHILE not(done) DO
    24  2     Begin
    24  2       getpage(lpage,done);
```

```
161    33  2        getpage(lpage,done);
162    42  2        getpage(rpage,done);
163    51  2        outputtriplepage;
164    54  2      end;
165    57  1  End. { of tpage }

PROC NAME        PSEC  PSIZE  LOCAL  STACK   CSEC  CSIZE  DEBUG
   0 TPAGE          7     59   8121     11      8      0      0
   1 GETC           1     60      0     15      2      0      0
   2 GETLINE        2     83      3     10      3      0      0
   3 GETPAGE        3    248    138     13      4      0      0
   4 OUTPUTTR       4    412      8     15      6     11      0
                         862   8270     64             11

165 Lines of source code compiled with no errors found
```

```
  1      0D  0 Program NDPAGE;
  2      0D  0 {NDPAGE is inspired by Kernigan and Plauger's book titled}
  3      0D  0 {  'Software Tools in Pascal'.                            }
  4      0D  0 { It simply takes a previously formatted file of text     }
  5      0D  0 { and builds a new file of dual column text pages.        }
  6      0D  0 { This particular program expects to read a file of text  }
  7      0D  0 { formatted with 54 columns per line at up to 66 lines    }
  8      0D  0 { per page. This version outputs page numbers, starting   }
  9      0D  0 { with the number as entered. It also strips the footer   }
 10      0D  0 { lines from the text, if there are any.                  }
 11      0D  0 { Page numbers are inserted on the 63rd line of each page}
 12      0D  0 {                                                         }
 13      0D  0 { Typical command line:                                   }
 14      0D  0 { DS9:Pascals <source >destination NDPAGE :page_number    }
 15      0D  0 {       If compiled into object code:                     }
 16      0D  0 { DS9:NDPAGE <source >destination :page_number            }
 17      0D  0 {                                                         }
 18      0D  0 { By E.W.Bollinger on September 15, 1982.                 }
 19      0D  0
 20      0D  0
 21      0D  0 CONST
 22      0D  0  ENDFILE = -1; { end of file marker }
 23      0D  0  NEWLINE = 13; { carraige return }
 24      0D  0  ENDSTR  =  0; { null }
 25      0D  0
 26      0D  0  MAXSTR = 132; { maximum string length }
 27      0D  0  MAXLINE = 56; { maximum text line length }
 28      0D  0  PAGELEN = 66; { maximum text page length in lines }
 29      0D  0
 30      0D  0 TYPE
 31      0D  0  character = -1..127; { ASCII plus ENDFILE }
 32      0D  0  string = array[1..MAXSTR] of character;
 33      0D  0  page = array[1..PAGELEN,1..MAXLINE] of character;
 34      0D  0
 35      0D  0 VAR
 36      0D  0  lpage,rpage: page; { left and right text page arrays }
37-14784D  0  done: boolean;
38-14785D  0  pn: integer; { number of pages counter }
39-14787D  0
40-14787D  0 FUNCTION GETC(var c: character): character;
 41      0D  1 { getc -- get one character from standard input }
 42      0D  1 VAR
 43      0D  1  ch: char;
 44     -1D  1 BEGIN
 45       0  2  IF (eof) then
 46       8  3    c:=ENDFILE
 47       9  3  ELSE IF (eoln) then
 48      21  4    Begin
 49      21  4      Readln;
 50      24  4      c:=NEWLINE
 51      25  4    End
 52      27  4  ELSE
 53      30  4    Begin
 54      30  4     Read(ch);
 55      35  4     c:=ord(ch);
 56      45  4    End;
 57      45  2  getc:=c;
 58      55  2 End; { of getc }
 59       0  1
 60       0  1 PROCEDURE GETPAGENUMBER(var pn: integer);
 61       0  1 { getpagenumber -- gets the page number from the parameter }
 62       0  1 { passed to it in SYSPARAM from the command line.          }
 63       0  1 var
```

```
 64      0D  1   i: integer;
 65     -2D  1   RETURN: char;
 66     -3D  1 Begin
 67       0  2   RETURN:=chr(NEWLINE);
 68       6  2   i:=0;
 69       8  2   While (sysparam[i]<>' ') and (i<79) DO
 70      32  3         i:=succ(i);
 71      38  2   sysparam[i]:=RETURN;
 72      51  2   i:=1;
 73      53  2   IF sysparam[0]<>RETURN Then
 74      67  3      i:=trunc(cnvtreal(sysparam));
 75      77  2   pn:=i;
 76      80  2 End; { of getpagenumber }
 77       0  1
 78       0  1 FUNCTION GETLINE(var s: string; maxsize: integer): boolean;
 79       0  1 { getline - get a line of text from standard input }
 80       0  1 Var
 81      0D  1   i: integer;
 82     -2D  1  ch: character;
 83     -4D  1 Begin
 84       0  2   i:=1;
 85       4  2   Repeat
 86       4  2     s[i]:=getc(ch);
 87      27  3     i:=succ(i);
 88      30  3   Until (ch=ENDFILE) or (ch=NEWLINE) or (i=maxsize);
 89      45  2   IF ch=ENDFILE then { gone one too far }
 90      52  3      i:=pred(i);
 91      55  2   s[i]:=ENDSTR; {mark end of string}
 92      68  2   getline:=(ch<>ENDFILE);
 93      75  2 End; { of getline}
 94       0  1
 95       0  1 PROCEDURE OUTPUTDOUBLEPAGE;
 96       0  1 { Takes two inputted text pages and outputs both onto }
 97       0  1 { one page                                             }
 98       0  1 Var
 99      0D  1  i,n,x: integer;
100     -6D  1 Begin
101       0  2  i:=1;
102       4  2  While i<=PAGELEN Do
103      11  3   Begin
104      11  3    IF i=64 THEN
105      18  4      Begin
106      18  4        write('          ');
107      26  4        FOR x:=1 to MAXLINE-4 DO
108      39  5          write(' ');
109      57  4        write('PAGE ',pn:4);
110      72  4        pn:=succ(pn)
111      76  4      End
112      79  4    ELSE
113      82  4      Begin
114      82  4         IF ((lpage[i,1]<>ENDSTR) and (rpage[i,1]<>ENDSTR)) or

115     129  4            ((lpage[i,1]=ENDSTR) and (rpage[i,1]<>ENDSTR)) or
116     177  4            ((lpage[i,1]<>ENDSTR) and (rpage[i,1]=ENDSTR)) THEN
117     228  5          Begin
118     228  5            write('          ');
119     236  5            n:=1;
120     238  5            WHILE (lpage[i,n]<>ENDSTR) and (n<MAXLINE) Do
121     278  6             Begin
122     278  6              write(chr(lpage[i,n]));
123     309  6              n:=succ(n)
124     311  6             End;
125     315  5            IF n<MAXLINE Then
126     322  6             Begin
127     322  6              FOR x:=n TO MAXLINE Do
128     333  7                write(' ');
129     351  6             End;
130     351  5            write('     ');
131     359  5
132     359  5            IF i<>3 THEN
133     365  6             Begin
134     365  6                n:=1;
135     367  6                While (rpage[i,n]<>ENDSTR) and (n<MAXLINE) Do
136     403  7                 Begin
137     403  7                  write(chr(rpage[i,n]));
138     434  7                  n:=succ(n)
139     436  7                 End;
140     440  6              end;
141     440  5         End;
142     440  4     End;
```

```
143   440  3     writeln;
144   443  3     i:=succ(i);
145   446  3    End;
146   449  2   End; { of outputdoublepage }
147     0  1
148     0  1 Procedure GETPAGE(var pg: page; var done: boolean);
149     0  1 { getpage - gets one page of text from standard input }
150     0  1 Var
151    0D  1  i,n: integer;
152   -40  1  s: string;
153 -268D  1 Begin
154     0  2  FOR i:=1 TO PAGELEN DO
155    17  3   begin
156    17  3     IF (not(done)) and (getline(s,MAILINE)) then
157    35  4      begin
158    35  4       n:=1;
159    37  4       IF (s[n]<>NEWLINE) and (s[n]<>ENDSTR) then
160    77  5        begin
161    77  5         REPEAT
162    77  5           pg[i,n]:=s[n];
163   120  6           n:=succ(n)
164   122  6         UNTIL (s[n]=NEWLINE) or (s[n]=ENDSTR) or (n=MAILINE);
165   160  5        end;
166   168  4       IF n<MAILINE then
167   175  5          pg[i,n]:=ENDSTR
168   196  5       ELSE
169   201  5          pg[i,n-1]:=ENDSTR;
170   226  4       end
171   226  4     ELSE

172   229  4       Begin
173   229  4         pg[i,1]:=ENDSTR;
174   247  4         pg[i,2]:=ENDSTR;
175   266  4         done:=true
176   267  4       end;
177   270  3   End;  { of for next loop }
178   284  2  End; { of getpage }
179     0  1
180     0  1 { Main program }
181     0  1  Begin
182     0  1    pn:=1;
183     6  1    getpagenumber(pn);
184    12  1    done:=false;
185    17  1    WHILE not(done) DO
186    24  2     Begin
187    24  2       getpage(lpage,done);
188    33  2       getpage(rpage,done);
189    42  2       outputdoublepage;
190    45  2     End;
191    48  1  End. { of NDPAGE program }

PROC NAME      PSEC  PSIZE  LOCAL  STACK   CSEC  CSIZE  DEBUG
  0 NDPAGE        9     50  14787      9     10      0      0
  1 GETC          1     56      1     15      2      0      0
  2 GETPAGEN      2     81      3     16      3      0      0
  3 GETLINE       3     77      4     13      4      0      0
  4 OUTPUTDO      4    450      8     15      6     32      0
  5 GETPAGE       7    286    270     13      9      0      0
                      1000  15073     81             32

191 Lines of source code compiled with no errors found

  1      0D  0 Program DPAGE:
  2      00  0 { DPAGE is inspired by Kernigan and Plauger's book titled}
  3      0D  0 {  'Software Tools in Pascal'.                           }
  4      00  0 { It simply takes a previously formatted file of text    }
  5      0D  0 { and builds a new file of dual column text pages.       }
  6      00  0 { This particular program expects to read a file of text }
  7      0D  0 { formatted with 54 columns per line at up to 66 lines   }
  8      00  0 { per page. This version does not strip header or footer }
  9      0D  0 { lines nor does it print page numbers.                  }
 10      0D  0 {                                                        }
 11      0D  0 { Typical command line:                                  }
 12      0D  0 {      Pascals <source >destination Dpage                }
 13      0D  0 {  or if compiled into native code:                      }
 14      0D  0 {      Dpage <source >destination                        }
 15      0D  0 {                                                        }
 16      0D  0 { By E.M.Bollinger on September 6, 1982.                 }
 17      0D  0
 18      0D  0
 19      0D  0 CONST

 20      0D  0  ENDFILE = -1; { end of file marker }
 21      00  0  NEWLINE = 13; { carraige return }
 22      0D  0  ENDSTR  =  0; { null }
 23      0D  0  SPACE = 32; { space character }
 24      00  0
 25      0D  0  MAXSTR = 132: { maximum string length }
 26      0D  0  MAILINE = 56; { maximum text line length }
 27      00  0  PAGELEN = 66; { maximum text page length in lines }
 28      0D  0
 29      00  0 TYPE
 30      00  0  character = -1..127; { ASCII plus ENDFILE }
 31      0D  0  string = array[1..MAXSTR] of character;
 32      0D  0  page = array[1..PAGELEN,1..MAILINE] of character;
 33      00  0
 34      0D  0 VAR
 35      0D  0  lpage,rpage: page; { left and right text page arrays }
36-14784D  0  done: boolean;
37-14785D  0
38-14785D  0 FUNCTION GETC(var c: character): character;
 39      0C  1 { getc -- get one character from standard input }
 40      0D  1 VAR
 41      0B  1  ch: char:
 42     -1D  1 BEGIN
 43       0  2  IF (eof) then
 44       8  3    c:=ENDFILE
 45       9  3  ELSE IF (eoln) then
 46      21  4    Begin
 47      21  4      Readln;
 48      24  4      c:=NEWLINE
 49      25  4    End
 50      27  4  ELSE
 51      30  4    Begin
 52      30  4     Read(ch);
 53      35  4     c:=ord(ch);
 54      45  4    End;
 55      45  2  getc:=c;
 56      55  2 END; { of getc }
 57       0  1

 58       0  1 FUNCTION GETLINE(var s: string: maxsize: integer): boolean;
 59       0  1 { getline - get a line of text from standard input }
 60       0  1 Var
 61      0D  1   i: integer;
 62     -2D  1  ch: character;
 63     -4D  1 Begin
 64       0  2   i:=1;
 65       4  2   Repeat
 66       4  2     s[i]:=getc(ch);
 67      27  3     i:=succ(i);
 68      30  3   Until (ch=ENDFILE) or (ch=NEWLINE) or (i=maxsize);
 69      45  2   IF ch=ENDFILE then { gone one too far }
 70      52  3     i:=pred(i);
 71      55  2   s[i]:=ENDSTR; {mark end of string}
 72      68  2   getline:=(ch<>ENDFILE);
 73      75  2 End; { of getline}
 74       0  1
 75       0  1 PROCEDURE OUTPUTDOUBLEPAGE;
 76       0  1 { Takes two inputted text pages and outputs both onto }
 77       0  1 { one page                                            }
 78       0  1 Var
 79      0B  1  i,n,x: integer;
 80     -6B  1 Begin
 81       0  2  i:=1;
 82       4  2  While i<=PAGELEN Do
 83      11  3   Begin
 84      11  3    IF ((lpage[i,1]<>ENDSTR) and (rpage[i,1]<>ENDSTR)) or
 85      58  3        ((lpage[i,1]=ENDSTR) and (rpage[i,1]<>ENDSTR)) or
 86     106  3       ((lpage[i,1]<>ENDSTR) and (rpage[i,1]=ENDSTR)) THEN
 87     157  4     Begin
 88     157  4       write('          ');
 89     165  4       n:=1;
 90     167  4       WHILE (lpage[i,n]<>ENDSTR) and (n<MAILINE) Do
 91     203  5        Begin
 92     203  5         write(chr(lpage[i,n]));
 93     234  5         n:=succ(n)
 94     236  5        End;
 95     240  4       IF n<MAILINE Then
 96     247  5        Begin
 97     247  5         FOR x:=n TO MAILINE Do
 98     259  6           write(' ');
 99     277  5        End;
```

```
100   277  4        write('      ');
101   285  4
102   285  4        n:=1;
103   287  4        While (rpage[i,n]<>ENDSTR) and (n<MAXLINE) Do
104   323  5         Begin
105   323  5          write(chr(rpage[i,n]));
106   354  5          n:=succ(n)
107   356  5         End;
108   360  4      End;
109   360  3     writeln;
110   363  3     i:=succ(i);
111   366  3    End;
112   369  2   End; { of outputdoublepage }
113     0  1
114     0  1 Procedure GETPAGE(var pg: page; var done: boolean);
115     0  1 { getpage - gets one page of text from standard input }
116     0  1 Var
117    0D  1   i,n: integer;
118   -4D  1   s: string;
119 -2680  1 Begin
120     0  2  FOR i:=1 TO PAGELEN DO
121    17  3   begin
122    17  3     IF (not(done)) and (getline(s,MAXLINE)) then
123    35  4      begin
124    35  4       n:=1;
125    37  4       IF (s[n]<>NEWLINE) and (s[n]<>ENDSTR) then
126    77  5        begin
127    77  5         REPEAT
128    77  5           pg[i,n]:=s[n];
129   120  6           n:=succ(n)
130   122  6         UNTIL (s[n]=NEWLINE) or (s[n]=ENDSTR) or (n=MAXLINE);
131   168  5        end;
132   168  4       IF n<MAXLINE then
133   175  5          pg[i,n]:=ENDSTR
134   196  5       ELSE
135   201  5          pg[i,n-1]:=ENDSTR;
136   226  4       end
137   226  4     ELSE
138   229  4       Begin
139   229  4         pg[i,1]:=ENDSTR;
140   247  4         pg[i,2]:=ENDSTR;
141   266  4         done:=true
142   267  4       end;
143   270  3   End;  { of for next loop }
144   284  2  End; { of getpage }
145     0  1
146     0  1 { Main program }
147     0  1  Begin
148     0  1    done:=false;
149     7  1    WHILE not(done) DO
150    14  2     Begin
151    14  2       getpage(lpage,done);
152    23  2       getpage(rpage,done);
153    32  2       outputdoublepage;
154    35  2     End;
155    38  1  End. { of DPAGE program }

PROC NAME      PSEC  PSIZE  LOCAL  STACK   CSEC  CSIZE  DEBUG
   0 DPAGE        8     40  14785     11      9      0      0
   1 GETC         1     56      1     15      2      0      0
   2 GETLINE      2     77      4     13      3      0      0
   3 OUTPUTDO     3    370      8     15      5     17      0
   4 GETPAGE      6    286    270     13      8      0      0
                       829  15068     67            17

155 Lines of source code compiled with no errors found
```

TURBO

I needed more memory. I have an application which requires fast access to a data array. 64k just isn't enough memory on my OS-9* level one system to keep everything going. Level one can't handle more than 64k. I had to have instant access to the data which was now nearly 16k all by itself, but I also wanted to use the system for other things while the application was running. Level two OS-9 seemed to be the only answer.

I found an alternative. A conversation with Jerry Kopple at AAA Chicago Computer Center lead me to consider the Computer Excellence 256k DRAM board. My Elektra* CPU-8/9* board doesn't have a dynamic address translator. I can't directly address 256k even if I had level two. I didn't really need level two any way I just needed to get to my data quickly. If I didn't need such quick access I could use a disk file. So what I really needed was not more memory but a very fast disk. The answer is a program which "looks" like a disk drive to OS-9 but with access that is just as fast as memory. There are some virtual disk programs around, but they also require a CPU board with resident DATs.

The board from Computer Excellence solves the problem. As the very complete documentation states, the Computer Excellence 256k DRAM board is built up on a double sided glass epoxy PC board with access to between one and four banks of dynamic address translators (DATs). It can accommodate several combinations of the currently available 41xx type dynamic RAMs including the new 256k chips. With 32 4164s the DATs control the placement of 64 4k pages. Any 16 of these blocks can be accessed at a particular time as part of the processor's 64k memory. Unfortunately a program which makes the memory board look like a disk drive did not exist. I liked the board itself though and took a chance that I would be able to write the virtual disk program. Before I went to OS-9 I had a SWTPc 6800* system. I have written 68xx assembler for five or six years now and expected the virtual disk program to be a good way to learn OS-9 calling conventions.

The result is the accompanying program. VDSK took some time to write but was not difficult. I had a disk driver program as an example and decided to make the program resemble a floppy. The OS-9 RBFman interface makes it possible to use not only all disk access system calls but I could even use the FORMAT program supplied with my floppy drivers for initializing the virtual disk in memory. With 800 extra disk sectors at my disposal I even have room for all of the system CMDS directory which normally resides on drive zero. The data file can be read about 100 times faster with the virtual disk than from a floppy. The system commands are not accelerated to the same extent because the loader computes a check sum for a load module which takes time. In general, data file access is instant and a print file can be read from the virtual disk to the printer without the slightest pause from a terminal running at 9600 baud at the same time.

I think the program explains itself. I would welcome comments and questions. These should be sent to me at the following address:

O E Groves 10207 Gillette Lenexa Kan. 66215

* OS-9 is a trademark of Microware systems Corp; Elektra and CPU-8/9 are trademarks of AAA Chicago Computer Center; SWTPC 6800 is a trademark of South West Technical Products Corp.

```
*                    *** TURBO ***
*
*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  VIRTUAL DISK SIMULATOR FOR THE COMPUTER EXCELLENCE MEMORY BOARD
*
*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
*       This program simulates a disk drive in memory.  Using the
*  onboard DAT of the Computer Excellence 256k board,  this virtual
*  disk simulator provides 800  256 byte sectors (less sector bit
*  map) to use as a super fast disk drive.
*      This driver program provides an OS9 level one interface to
* the Random Block File Manager (RBFMan) which looks exactly like
* a single sided drive with 800 sectors on it.  Logical sector
* numbers are translated directly to physical 256 byte "sectors" in
* 4k physical page in the extended address space.
*
* Author:  DE Groves of KITTEWARE - a member company of E-LABO
*             Enterprizes, 10207 Gillette, Lenexa, Kn.  66215
*
*      To install the VDSK driver in an OS-9 level one system
* the following command sequence may be used:
*
*        LOAD /XX/VDSK.OBJ    (XX= drive containing the VDSK
*                              program)
*        LINK VDSK            (So the system can find it)
*
*        FORMAT /V0           (FORMAT will default to a 50 track
*                              16 sector/track SS floppy drive)
*
*     If you decide to make VDSK a permanent part of your system
* do this:
*
*        COPY /XX/VDSK.OBJ /D0/VDSK    ie. put a copy of VDSK
*                                      on your system disk.
*
*              - put a formatted disk in drive D1 -
*
*        OS9GEN /D1
*        /D0/OS9BOOT                   use regular boot file
*        /D0/VDSK                      add VDSK
*        (esc)                         end of file
*
*  The disk in drive D1 will now have a boot file which
*   automatically loads VDSK
*
*
*             * * * * *   W A R N I N G   * * * *
*
*      Execution of NMI while accessing the virtual disk will
* leave the DAT in an INDETERMINATE state. The initialization
* routine for the DAT must be executed BEFORE going on.
*
*            DEVICE DESCRIPTOR FOR "V0"
*
*      This is the information IOMan needs to get to the Virtual
* disk number 0
*
* VIRTUAL DISK device descriptor
*
 NAM V0
 TTL Device Descriptor for "V0"
 IFP1
 ENDC
*
*       *******
*
*   Virtual DISK device module
*
TYPE SET DEVIC+OBJCT
REV SET REENT+1
 MOD V0END,V0NAM,TYPE,REV,V0MGR,V0DVR
 FCB $FF mode
 FCB $0F
 FDB $FFFD device controller address
 FCB V0NAM-*-1
 FCB DT.RBF device type= RBF
*
* default path options
*
 FCB 0 drive number
 FCB 0 step rate na
 FCB 0 device type ="5","std","floppy"
 FCB 0 density    ="single","single"
 FDB 50 # tracks
 FCB 1 num sides
 FCB 1 no verify
 FDB 16 sectors/track
 FDB 16 sectors on track 0
 FCB 1 no interleave
 FCB 1 sector/block
```

```
V0NAM FCS "V0" Device name
V0MGR FCS "RBF" File manager name
V0DVR FCS "VDSK" Device driver name
 EMOD
V0END EQU *
*
*   THIS IS THE END OF THE VIRTUAL DISK DEVICE DESCRIPTOR
*   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
*  THIS IS THE BEGINING OF THE DEVICE DRIVER
*
 NAM VDSK
 TTL Device driver for Virt. DISK
 IFP1
 USE /D1/ELEKTRA_DEFS/OS9DEFS get system definitions
 USE /D1/ELEKTRA_DEFS/OS9RBFDEFS.2 random block defs
 ENDC
DRVCNT SET 1 only one virtual drive defined
*
*       ********    ram space definition   ********
*
 ORG DRVBEG put in place in static storage
 RMB DRVMEM*DRVCNT a table for each drive
CURTBL RMB 2 table number for this access
CURDRV RMB 1 drive number for this access
V.FREZ RMB 1 freeze DD.INFO
VDSKST EQU . total ram reserved
*
*       ********   MODULE HEADER   *******
*
TYPE SET DRIVR+OBJCT
REV SET REENT+1
 MOD DSKEND,DSKNAM,TYPE,REV,DSKENT,VDSKST
 FCB $FF access to public
DSKNAM FCS "VDSK"
 FCB 1 rev number
*
*         branch table
*
*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  ENTRY POINT
*
DSKENT LBRA INVDSK
 LBRA VDSKRD
 LBRA VDSKWT
 LBRA VDSKGS
 LBRA VDSKPS
 LBRA VDSKTM
*
*  INITIALIZATION
*
INVDSK PSHS X
 LEAX DRVBEG,U point to drive table
 LDB #DRVCNT
 STB V.NDRV,U set max drives
 LDD #800
 STD DD.TOT+1,X
 STB V.TRAK,X set to high count
 CLR V.FREZ,X
 PULS X,PC go home
*
*
*  READ SECTOR
*
*        input -  B=MSB of logical sector number
*                 X=rest of sector number
*                 Y=path descriptor
*                 U=global storage
*
*    The Virtual disk is read by computing a 4k page/track number
*  in extended memory and a 12 bit offset into the page for a
*  256 byte "sector."  The computed page is swapped into the
*  program's address space at page zero.  In the case (very
*  rare) that the program's buffer lies on page zero, page
*  one (1000-1fff) will be used.
*
VDSKRD LEAX ,X read sector zero?
 BEQ RDZERO special processing
DOREAD  BSR RDSEC
 BCS RDNG
 CLRB
RDNG RTS go home
**

RDSEC BSR GTADDR get page(trk) & offset(sectr)
 BCS RDNG
 PSHS CC,D,DP,X,Y,U save working regs
 LDX PD.BUF,Y pick up buffer address
 EXG X,D .  D= buffadr: X= sector #
 ANDA #$F0 where is the buffer?
 BNE RDK.0 buffer not in page 0
 INC 4,S put vdisk on page 1
RDK.0 LDB 4,S recover track number
 LDY PD.BUF,Y get buffer addr
 LDU V.PORT,U get controller address
 TFR A,DP I needed a reg.: save mempage
 ORCC #$50 no interrupts while page 0 gone
 STB A,U swap in page (track)
 CLRB multiply A x $1000
 LSLA . to get logical addr
 LSLA
 LSLA
 LSLA
 LEAX D,X add addr to offset
RDLP LDA ,X+ frm sec 0000-0F00 or 1000-1F00
 STA ,Y+ to buffer
 DECB transfer 256 bytes
 BNE RDLP
 TFR DP,A
 STA A,U swap page back
 CLR 2,S
 PULS CC,D,DP,X,Y,U,PC go home
* * * * * * * * * * * * * * * * *
RDZERO BSR RDSEC
 BCS RDNG
 PSHS X,Y
 LDX PD.BUF,Y transfer volume info
 LEAY DRVBEG,U to drive table
 LDB #DD.SIZ-1
RDZER1 LDA B,X
 STA B,Y
 DECB
 BPL RDZER1
 CLRB
 PULS X,Y,PC go home
*           *********
GTADDR TSTB 2 byte sector numbers only
 BNE ADERR
 PSHS B
 ANDCC #$FE clr carry bit
 TFR X,D . D=logical sector (0000-0F00)
 LSRA divide logical sector # by 16
 RORB
 ROR ,S .   put remainder on stack
 LSRA
 RORB
 ROR ,S
 LSRA
 RORB
 ROR ,S
 LSRA .  A=0
 RORB .  B=track # (0-31)
 ROR ,S .  ,S=offset x 16
 ADDB #$E . pages 0-D in use already
 TFR B,X .  X=track # now (E-3F)
 PULS B .  B=offset (sector # x 16)
 LSRB
 LSRB
 LSRB
 LSRB
 EXG A,B A=sector# (0-F) B=0
 RTS
ADERR COMB
 LDB #E$SECT
WTNG RTS
*  WRITE SECTOR
*
*             input  -  B=MSB of logical sector number
*                       X=rest of logical sector number
*                       Y=path descriptor
*                       U=global storage
```

```
*
*    Writing a sector is the same as reading except that
*  transfer is from the buffer to the virtual disk sector.
*
VDSKWT BSR GTADDR get page(trk) & offset(sectr)
 BCS WTNG
 PSHS CC,D,DP,X,Y,U save working regs
 LDX PD.BUF,Y pick up buffer address
 EXG X,D .  D= buffadr: X= sector #
 ANDA #$F0 where is the buffer?
 BNE OK.0 buffer not in page 0
 INC 4,S put vdisk on page 1
OK.0 LDB 4,S recover track # (E-3F)
 LDY PD.BUF,Y get buffer addr
 LDU V.PORT,U get controller address
 TFR A,DP I needed a reg.: save mempage
 ORCC #$50 no interrupts while page 0 gone
 STB A,U swap in page (track)
 CLRB multiply A x $1000
 LSLA . to get logical addr
 LSLA
 LSLA
 LSLA
 LEAX D,X add addr to offset
WTLP LDA ,Y+ from buffer
 STA ,X+ to sect 0000-0F00 or 1000-1F00
 DECB transfer 256 bytes
 BNE WTLP
 TFR DP,A
 STA A,U swap page back
 CLR 2,S clrb
 PULS CC,D,DP,X,Y,U,PC go home
* * * * * * * * * * * * * * * *
*
*  PUT/SET STATUS
*
VDSKPS LDX PD.RGS,Y point to parameters
 LDB R$B,X what status are we putting?
 CMPB #SS.RST restore ?
 BEQ NOPER
 CMPB #SS.WTK write (format) ?
 BEQ NOPER
 CMPB #SS.FRZ freeze DD.INFO?
 BEQ FREZINF
 CMPB #SS.SPT set sectors/track ?
 BEQ NOPER
*
*  GET STATUS
*
VDSKGS COMB none of above or VDSKGS: error
 LDB #E$USVC get error code
 RTS
FREZINF LDB #$FF
 STB V.FREZ,U
NOPER CLRB na for this drive
 RTS
*
*  TERMINATE VIRTUAL DISK
VDSKTM CLRB
 RTS no action needed
*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  END VIRTUAL DISK device driver module
*
 EMOD
DSKEND EQU *
```

SINGLE BOARD COMPUTERS-6809

Single Board Computers
Sardis ST-2900 Update Report

When we started the series of reviews of single board 6809 computers, I realized that I would have to probably do some updates. That is exactly what I wanted. Product updates-upgrades-improvements-better values, what ever you want to call them, they all stand to benefit you, my readers. And that is what this thing has been all about!

So, I am delighted to report to you any and all improvements of these fine products. The newer generation of micro-computers have nipped us here and there, but we have something that no other group of owner/users have; we can get to the 'guts' of the thing. Also, we can build it ourself, if we so desire. Try to build an 8087 system, I mean - **COMPLETE.** Right you are, you CANNOT! But, you can build a very powerful, complete 6809 computer, right here out of the pages of 68 MICRO JOURNAL™. And these very same boards are part of that project. In addition there are the bare boards and other hardware advertised, in our pages, that let you have the greatest variety of micro-computer building blocks offered anywhere to that special breed of individual who still takes pride in, "I built it myself and **SAVED** money in the process."

In addition, you can buy some of the **worlds best micro-computers** right here out of the pages of 68 MICRO JOURNAL. Then you can expand to your hearts content, with just what you need, 'store bought' or 'roll your own'. No source anywhere offers you as muc !

Sardis ST-2900 Update

We received a complete set of the new and 'improved' documentation for this system after our original review had gone to press. Still not "Heath' quality but completely sufficient, and much improved.

No dot-matrix printer typesetting. No penciled overstrikes. No errors I could find. All diagrams and charts simple and easy to understand, and professionally done. Above all, **simple** instructions on adapting I/O direct addressing software such as STYLO™, DYNACALC™, RMS™ and SCREDITOR III™. These are the only software packages I can think of that need this special treatment. Once done they run the same as on any ot er 6809 system.

Below I will briefly outline the latest improvements:

1. The monitor has been changed to accommodate serial handshaking. Also, the "M" memory examine and change routine has been expanded.

2. The changes to the FLEX™ conversion package has resulted in improved utilities, such as FORMAT and DSKSET.

a. FORMAT: This utility which replaces the TSC newdisk routine has the following features:

a1. LOWER DENSITY - TSC format of 10 sectors per track single density and 18 sectors double density. Or the IBM(?) 3740 9 single, 16 double density.

a2. SWTPC FORMAT - this allows disks formatted with this option SWTPC compatibility. Therefore you have the choice of either the SWTPC or GIMIX formats in the event you swap programs on disk with other users. This has been a real hassle for some.

3. 35-40-80 track formatting now possible.

4. Some utilities have been added:

LOADO - this offset loads from disk to memory (but does not execute) a binary file. Offsetting is not necessary but then the TSC GET command would do.

PATCHES.OVR allows patching FLEX™, which has been a bear at times for some who wish to insert user code in front of the transfer address. FLEX normally stops loading when the transfer address is encountered at load time.

PRTSET - allows printer driver parameters to be changed - NC, nulls after C/R - NL, nulls after L/F - LM, left margin width - BR, printer port baud rate. Also the source code for all printer drivers (3) is included.

SYSGEN - creates a FLEX system file that boots direct from the monitor. It allows PATCHES.

BLIST - a LIST utility similar to the TSC version but it fills much more of memory with the cal ed text file, causing less wear and tear on the disk system.

CHECKSUM - a utility that reports the checksum of any disk file.

DCHECK - a utility to allow checking visually on the CRT screen the rotational sp d of all disk drives, also measures the delay time from 'motor on' signal to the 'ready' signal to the 1793 disk controller.

DSKSET - improved version that allows complete parameters for practically any type disk system, including the newer Shugart SA300 3 1/2 inch types.

OS9.CMD - this utility allows the user to call the Radio Shack version of OS-9™ from FLEX. A prompt is issued to insert your OS-9 disk. In about 30 seconds up pops the TAN Y OS-9 banner and you are in OS-9.

Some portions have not been completed. Ours is an evaluation sample. For instance the clock is not implemented yet, but will be probably by the time you read this. Also Sardis has licensed from D.P. Johnson SDISK and SFORMAT. These allow both CoCo and standard OS-9 disk formats to run on the system. Considering the price of TANDY OS-9 and Basic09, this makes for quite a savings. While not spending a lot of time with this version of OS-9 (we have several OS-9 systems, all larger, including the GIMIX III) I found little to fault, even with a 'Beta' test version. Also, because of the DUART used and the addressing space of the I/O, in OS-9 there is available 63.75K of RAM available. No 'holes' in the middle of the address map. This should be one of the 'better' OS-9 level 1 systems, due to the memory allocation.

Some enhancements being considered for the future are; booting from a 48 tpi disk in a 96 tpi drive. Expanded memory, and possibly a DAT configuration for extended memory applications.

All in all, as I said before, these little 'jewels' are great for those desiring a complete 64K FLEX system. And now OS-9 also.

As the others pass along to us the updates, upgrades and improvements of their systems, I will report them to you.

DMW

- - -

MICROKEY 4500

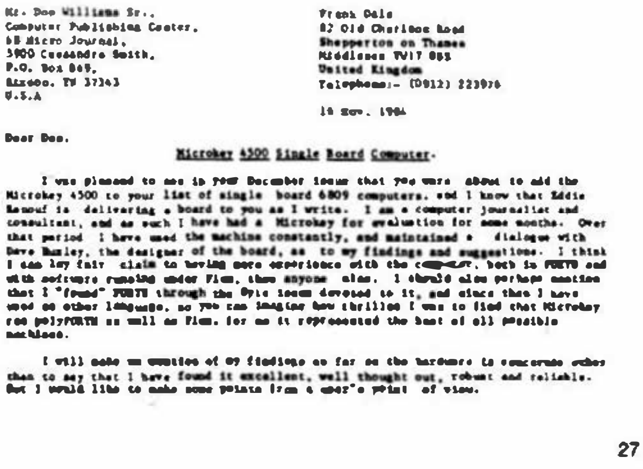
MICROKEY 4500

The Ukey 4500 is an entirely different type of SBC, as compared to the others. First, it has normally 128K bytes of RAM. Secondly, it sports many different CPU devices -W65SC802, W65SC816 6502, 65C112 and the 6809E. Thirdly, it also has fi er-optics I/O as well as hardwire. **And the color high speed, high resolution graphics are superior to most any microcomputer now available!!** In addition it can fall into the $1500 class, as the others do. However, fully loaded, it will exceed that price by a couple hundred dollars (special introductory price - see advertising). The normal price will be somewhat higher depending on USA distribution, shipping cost, import duties and all those other cost that get added on in the norm l course of business. But even at twice the introductory price it will still be a bargain!

ANOTHER VIEW

Below is a users comments of the Ukey 4500 as we received recently. Although we have a complete system inhouse, I thought you might like another view from an actual user, who has had somewhat more opportunity to evaluate the Ukey 4500 than we have so far.

---



Mr. Don Williams Sr.,
Computer Publishing Center,
68 Micro Journal,
5900 Cassandra Smith,
P.O. Box 849,
Hixson, TN 37343
U.S.A

Frank Dale
82 Old Charlton Road
Shepperton on Thames
Middlesex TW17 8BJ
United Kingdom
Telephone:- (0932) 223976

18 Nov. 1984

Dear Don,

Microkey 4500 Single Board Computer.

I was pleased to see in your December issue that you were about to add the Microkey 4500 to your list of single board 6809 computers, and I know that Eddie Renouf is delivering a board to you as I write. I am a computer journalist and consultant, and as such I have had a Microkey for evaluation for some months. Over that period I have used the machine constantly, and maintained a dialogue with Dave Huxley, the designer of the board, as to my findings and suggestions. I think I can lay fair claim to having more experience with the computer, both in FORTH and with software running under Flex, than anyone else. I should also perhaps mention that I "found" FORTH through the Byte issue devoted to it, and since then I have used no other language, so you can imagine how thrilled I was to find that Microkey ran polyFORTH as well as Flex, for me it represented the best of all possible machines.

I will make no mention of my findings as far as the hardware is concerned other than to say that I have found it excellent, well thought out, robust and reliable. But I would like to make some points from a user's point of view.

The system as I have it is configured for two Sony Micro Drives as drives zero and one, and one 5" 35 track single-density single-sided drive as drive 2. The Sony drives are magnificent, a very large capacity and a very fast data rate, it's just like having a hard disk on line. And the 5" drive makes transferring software from other Flex systems very simple. I have two monitors, one color and one black and white, and two keyboards, plus my daisy-wheel printer hooked in. I also have communication with other computers in the house via RS232.

I was aware very early on of the problems of re-configuring some of my Flex software to run on Microkey because of the way software like Dynacalc and RMS tended to address the Motorola ACIA directly, and Microkey uses a very different system. Part of Dave's design philosophy, the reason there is so little provision for user I/O, was that most noise problems are associated with I/O lines, and the next stage in the design was to develop RS232-driven fibre-optic communications on a separate tiny board in order to more suit the 6500 for control purposes in factory environments.

The decision was made to not only provide the fibre-optic capability, but to use two Motorola ACIA's so that one could either hook up the fibre or just plug in a separate terminal, or both. I have the prototype board on my Microkey at this moment, and it works beautifully.

I can run all TSC's Flex software using the Microkey alone. All the software which uses Flex for all of its communications works very happily with the color monitor and Microkey's keyboard. For the difficult software I plug in my CT-82, type "TERM" on Microkey's keyboard, and I am running the Microkey using the CT-82 as a separate ACIA-driven terminal.

Dave is now going to "hook" the ACIA back into the system, and then you will not even need to plug in the separate terminal, just direct communications to the ACIA and Microkey will behave as if you had plugged in a separate terminal. So you have the choice. Plug in a terminal and Microkey will work with it. Unplug the terminal and Microkey will behave as if it had a terminal on line as far as the software is concerned.

It seems to me to be the best of all possible worlds. Without terminal I have tested most of the major packages, all the TSC UTILITIES, TSC Pascal, Asmb, Edit, TP, Debug, Sort/Merge. I have also tested Lucidata Pascal, Lisp, SWT's Symbolic Disassembler, and the Windrush software like MACE and its family and PL/9. All of these run with little or no modification.

With the CT-82 plugged in I have tested Dynacalc, RMS, Basic09 and Extended Basic (of TSC), and John Alford's SCREDITOR III. All are well documented as to re-configuring for a different ACIA address which is often all that is needed. As readers will know, many of these programs come with configuration software which prompts for all the terminal and keyboard characteristics.

The only slight problem I had was with Dynacalc. That program is so User Friendly that it becomes a pain in the neck to re-configure it. I took the easy way out. I copied the version I had made for the CT-82 to Microkey's Sony drive and then changed the two bytes at $000F from $DF60 (SWT vector) to $F700 (Microkey vector).

Of course, in the time available, I cannot claim to have exhaustively tested all of the software I have listed, but I have it running and I have put test programs through all of it.

Very early on I brought up Talbot's tFORTH+ on Microkey. I would not recommend any non-FORTH users to try this because it requires a little understanding of FORTH. As purchased for my SWT computer, Talbot is configured for two 5" drives as zero and one, and two 8" drives as two and three. Obviously the requisite parameters have to be changed in order for it to compile from the Sony disks. Secondly, the screen output of Talbot expects SBUG to take care of the eighth bit of ASCII characters (FORTH uses that bit in name fields), and Microkey, with its full color and high resolution graphics, passes all eight bits to the screen. The definition of the FORTH word "ID." must be re-written so as to kill the eighth bit, and then it must be patched into the kernel, otherwise a dictionary dump will be difficult to read. The words which set up and communicate with a serial interface (I-SER ?SER OUSER and CISER) need to be re-written in order to be able to use Microkey's user serial interface, although they must also be retained in their original form in order to access the two ACIA's on the communications board.

The screen editor has to be reconfigured in order to make use of different control keys on the keyboard, although the fig line editor, of course, will work without any changes. And a few primitives and high-level words need to be written in order to use Microkey's Rom routines for color graphics. Finally the "80" screen, screen 43, has to be modified in order to incorporate the changes.

All the foregoing is trivial and very easy to do provided you understand fig-FORTH and the particular implementation you wish to host on Microkey. One advantage in using Talbot over against polyFORTH is that Talbot uses disks formatted in Flex format whereas polyFORTH uses a quite different format. Talbot can read and write Flex text files as well as the normal FORTH virtual memory blocks, and it can use disks partitioned to be part virtual and part Flex. So it can take files of data output from RMS or Dynacalc, use that data in an application, and then send new data back into text files. If I am writing about FORTH, for instance, I can send a listing of FORTH screens to a Flex file where TSC's TP can incorporate it into the body of my article.

I have just completed a pseudo floating point in FORTH for Microkey. It looks and acts like floating point but is in fact entirely integer and scaler maths, including sine and cosine routines, which is used in a 3D color graphics package which uses matrix tranformations in order to produce rotations, magnifications etc. The results are very fast and impressive, real-time color animation on Microkey's Dual 6809.

The Microkey support environment, read in from ROM at power-up, includes a host of subroutines which the user can call to invoke screen, keyboard and system functions. These are particularly easy for FORTH to access, the user has only to push any required parameters onto the stack and then execute an indirect JSR. Microkey function words can very easily be written to change to and from color and mono, change text color and graphics pixel color, draw lines between XY coordinates, plot points etc. Graphics and text can be freely intermixed on the screen as can the colors, s the user can create some really dazzling displays.

Another function, which I have tested but not really used in an application as yet, will switch software from one RAM bank to the other. Very useful because you can use it as a sort of "RAM DISK" in order to switch an application over to the other RAM at some point in its execution, invoke and execute a second application, and then switch back the first and continue its execution.

I have been looking for some time for a suitable single board 6809 with which to supplement my ageing SWT box. I have now decidedly found it! What is more, I am putting my money where my mouth is. This Microkey is mine. I have transferred all my work onto it, the SWT hasn't been switched on for three months, and Microkey has performed faultlessly. Don't be put off if you are not a FORTH man, this is a very very good Flex computer, good as a computer and good as a development and control system, but just add the FORTH capability as well and it has to be tried to be believed.

Finally, if you would like listings or other details, just ask. And anyone out there who has a problem with this board, just write. I'd be happy to help.

Yours sincerely,

Frank Dale

While the system is capable of running both the 6502 and the 16 bit version the 65SC816 CPU, we will be devoting most of our attention to the 6809 aspects of the Ukey 4500. Therefore the following 6502-65SC816 discussion is far from a complete overview of their capabilities. However, the basic system remains the same, it is usually supplied (and priced) for each CPU or combination of CPUs.

The 6502 runs both Forth-79 and Fig Forth. Built in editors and assemblers are part of both Forths. As a control language Forth is ideal. In fact Forth was developed as a control language for one of the worlds largest telescopes. While the use of 'reverse Polish' has never quite agreed with me, I know many professional and hobby programmers who would use nothing but Forth. It is an excellent language and is probably one of the most easily expandable HLLs available today.

The 16 bit version (65SC816) can address 512K RAM configured as two 256K banks. Clock speeds for the 6809E and the 16 bit CPU are approximately two Mhz, but the 6502 runs at a slower clock speed.

The color graphics running under Forth are so dazzling that words cannot describe them! The speed and resolution is some of the best we have ever seen running on ANY microcomputer!

The 809E Version

The 6809E system runs FLEX™, Talbot's tForth+ and polyForth. Now talk about your color graphics speed and resolution, even better!! Also you can run all the other popular FLEX software with little change (see Frank Dale's comments above).

The FLEX system also allows you to run all those other applications - business, accounting, OEM and software development, spread sheet, word processing, etc. - **including** high resolution, fast speed color graphics. Text and graphics mixed if desired.

Of all the SBCs we have reviewed and tested todate, this is the most complex, and therefore, this review cannot cover all of the advanced features availab e on the Ukey 4500. However, below is a brief overlook of the specs:

DMA all devices except the 6502, from expansion port.

RAM - 128K, two 64K banks, expandable to 512K with W65SC816.

REFRESH by video controller, RAM first cycle, CPU second.

EPROM - 32K, two 27128, 16K, two 2764. EPROMs can be switched out completely or replaced by external plug-in EPROMs.

SERIAL interfaces - TX,CTS,DCD,DSR,DTR,RTS provided at RS232 port. Connector 25 pin D plug. 16 software programmable rate up to 19,200. Device is a 6551 ACIA.

PARALLEL interfaces - two, standard TTL, 15 pin D plug. Device is a 6522.

KEYBOARD interface - two IBM type keyboard interfaces provided. Connectors are 180 degree 5 pin DIN sockets.

CASSETTE interface - Apple protocol only as of this date.

EXPANSION port - All bus control, data and power signals are provided on a 50 pin IDC plug.

SCREEN 1 - video output is composite monochrome (B/W), resolution is 640X200 pixels, low resolution mode, 1280X200 high resolution mode. Connector is RCA type phono plug.

TTL COLOR video - RGB at TTL levels plus separate TTL sync (H/V), connector 6 pin DIN plug. 8 colors available in color mode. In monochrome this output is same a Screen 1. Same resolution as Screen 1 for low level in color, high level is monochrome.

ANALOG color - RGB at 1 volt peak analog levels. with composite TTL level sync. Connector 7 pin DIN plug. 16 colors possible. resolution same as TTL color.

DISK DRIVES - provisions and connectors for Sony type 3.5 inch drives, Epson 3.5 and any 5.25 inch drive. Single and/or double density. Data rates of 125K, 250K and 500K bit/sec possible. A total of four drives may be on line and connected. Drive types may be mixed.

AUTOMATIC 'boot' search all drives for system boot program, not restricted to drive 0.

Expansion Port

An expansion port has been provided to allow extended or extra I/O ** see fiber-optics discussion **, memory or CPUs to be added to the Ukey system bus. DMA is allowed provided it IS NOT the 6502 CPU in control. The bus is unbuffered, large numbers of external devices or long runs of cable will require buffering. Even though the 6502 used is the 3 Mhz version, much better timing margins are possible using the 68B09E, and other CPUs.

External devices see the bus as a 100pf load with about 100uA leakage current. For data lines the load is

1 LSTTL and about 100pf.

The bus is capable of driving 30pf and 3 LSTTL loads. Cable lengths are critical.

Mo itor

The monitor has most of the popular monitor functions. In addition there is the command 'F' to run Forth, 'H' set high resolution graphics, 'I' set inverse video, 'J' set monochrome with software scroll. 'K' demo interlace video (useful for OEM applications - 16 color with 640X384 pixel display), 'L' load RAM from communications port, 'N' set color video, 'S' send memory content to serial port, 'T' terminal mode (enable system to act as a terminal and 'Z' set color - a 2 digit hex number sets the following text or graphics to the specific color, if both digits are not equal then two different colors will be used.

Documentation

As with the other SBCs, the documentation is not 'Heath' quality. For the Ukey 4500 the documentation is sufficient and cover both monochrome and color generation of both text and graphics. An especially nice feature is that all graphics and any text may be displayed at the same time. Because of the extensive use of Forth, Forth is covered in extra detail. However, the user should be somewhat familiar with Forth to gain the full utilization of this system.

Included in the documentation is info concerning the memory mapping schemes, in different modes. The one thing missing (at least from our system) is the inclusion of diagrams, parts layouts and parts list, all of which could be vital in event service is required.

Speaking of service, it should be a remote point for the system is top quality and has been completely tested and burned-in prior to shiping. Our inspection of the system shows a lot of attention to even the small details. And it has been in production and use in Europe for some time, in various configurations.

Optical-Fiber Interface

There is available a fiber-optics interface for OEM and control operations in industrial environments. It consist of a small board thet is driver from the serial port and eliminates the problems normally encounte ed in situations where long cables would have generated noise problems. (see Frank Dales review above) As noted above two additional ACIA's become available for system use.

Note should be made of those programs that address an ACIA directly. As with the Sardis system (reviewed earlier - and which this article is being written, using Stylo which addresses direct to an ACIA).

Conclusion

After using the Ukey 4500 it becomes somewhat mundane to go back to a B/W CRT display. You would be surprised as to how much time can be spent just playing with the high resolution color graphics. Having all this much power and versatility readily at hand makes thinking up new things to do lots of fun as well as being useful and productive.

If you order the complete system, you will find the quality of the keyboard (IBM type) and enclosure too quality and very professional. The addition of a CRT terminal, Color monitor or monochrome monitor makes a complete system, and at a price that is practically unbeatable, for all the features included. So either way, SBC or full system is a bargain at these intro prices.

Price as of this writing:

Full system, including 128K RAM, three drives (two 3.5 Sony or Epson and 5.25 standard, keyboard and enclosure w/power supply: $1,899.00

Fully burned-in and tested SBC - 128K RAM, less drives, keyboard and enclosure: $450.00.

Please note that shipping and taxes are extre.

For additional information contact:

MICROKEY Limited
98a, St. James Street
Brighton, Sussex, England
Tel 0273-672911

- - -

LOCAL

It is a utility called "LOCAL" which uses the SwTPC Dynamic Address Translator (DAT) to store command files in your system's extra memory and links them into FLEX's User Command Table so that they are copied directly from memory when invoked. System requirements are a DAT, 2K bytes of RAM from $E800 to $EFFF and, of course, enough extra memory.

The source of this program is written in WHIMSICAL, which is now available through '68' Micro Journal. We have found that a great advantage of WHIMSICAL is the ability to break programs into modules. These modules can be developed and compiled separately and over the last two years we have put together a library of them.

We have found that the WHIMSICAL language has been well designed and has a very consistant and intuative syntax (unlike many languages e.g. "C") thus making programs easy to write and just as easy to understand six months later.

We hope that the enclosed program will demonstrate the e features as well as being as useful for you as it is for us.

By the way, since Ron Anderson's review of WHIMSICAL in the Sept. '83 issue of '68' Micro Journal, REAL numbers (otherwi e known as floats) have been added to the compiler.

Regards,
(Mark Armstrong)

Mark Armstrong
12 Saltburn Road
Takapuna 9
Auckland
NEW ZEALAND
Ph: 498-843

```
% Make Command Files Reside "LOCALLY" in RAM (File E4LOCAL) 17 JUN 84
% ---- ------- ----- ------ ---------- -- ---
% Compiled by WHIM VER 1.5:54

% by M G Armstrong
% 12 Saltburn Road, Milford
% Auckland 9,   New Zealand

% System Requirements:
% 1) SwTPC compatible DAT
% 2) 2K bytes RAM from $E800 to $EFFF

% Command Syntax:
% +++LOCAL[,<file spec>[,<file spec>]][+<options>]

% File specs default to System drive with CMD extension.
% Options are   C  Catalogue the User Command Table
%               I  Provide extra information on memory usage
%               U  Unhook Local (removes User Command Table)

` STACK=([$CC2B])                     Set STACK according to Memory End
` ORIGIN=($1000)                      Use $0000-$0FFF for DATing in blocks
` VERSION 4,'.E4LOCAL, by M G Armstrong'

begin
  BYTE  CmdTableStart:= $E800,
        CmdTableEnd:=   $EAFF,
        ExecTableStart:=$EB00,
        ExecTableEnd:=  $EBFF,
        ProgramID($EFFE);
        proc EscapeToEOL=forward;
```

```
module Error= code from "E4ERROR";     % Error Handler
module Parse= code from "E4PARSE";     % Command Line Parser
module Block= code from "E4BLOCK";     % DAT Block Control
module ExTbl= code from "E4EXTBL";     % Execution Table
module CmdTbl=code from "E4CMDTBL";    % Command Table
module Load=  code from "E4LOAD";      % File Loader

do begin
| Err:=false;                          % Reset error flag
| Parse;                               % Next item from command line
| case ParseType of
| begin
|   1: LoadFile;                       % Load file into RAM and generate load map
|      CmdTable;                       % Enter file name into Command Table
|      ExecTable;                      % Enter file data into Execution Table
|   2: if Catalog then                 % Options (handled by Parse)
|      begin
|        CatCmdTable;
|        Catalog:=false;
|      end;
|      if UnhookLocal then
|      begin
|        Emit2(CmdTableStart, $0000);  % Destroy Command Table
|        CmdTableStart:=$CC12;         % Set variable to addresss of UCT
|        Emit2(CmdTableStart, $0000);  % Unhook User Command Table
|        Emit2(ExecTableStart, $0000); % Destroy Execution Table
|        ProgramID:=$0000;             % make LOCAL reinitialise

|        ParseType:=4;                 % skip to end of program
|      end;
|   3: ;                               % End of Command Line
|   else: Error($83);                  % Invalid command line
| end;
end until ParseType>=3;
RestoreBlocks;                         % Restore original DAT configuration
if ParseType<>4 then
begin
| CloseCmdTable;                       % Close the Command Table
| if ExtraInfo then
| begin
|   write "^M^JTotal Blocks used: ",     TotalUsed;
|   write "^M^JFree Blocks remaining: ", FreeLeft;
|   write "^M^JTotal Bytes Stored: ",    TotalBytes;
| end;
end;
EscapeToEOL;                           % Ensure TTYEOL recognised

end.

% ======================================================================


*TITLE="Error Handler" (File E4ERROR) 30 APR 84

module Error=
begin
public
  BOOL Err;
  proc Error(BYTE ErrNo);
external
  DBYTE ProgramID($EFFE);
  proc EscapeToEOL;
private
  proc Error(BYTE ErrNo)=
% ---- -----
  begin
    CHAR Answer;
    if NOT Err then
    begin
    | Err:=true;
    | write "^M^J";
    | case ErrNo of
    | begin
    |   $80:  write "Invalid drive number";
    |   $81:  write "Invalid file name";
    |   $82:  write "Drive specified twice";
    |   $83:  write "Invalid command line";
    |   $84:  write "Invalid options";
    |   $85:  write "File won't fit";
    |   $86:  write "Not a binary file";
    |   $87:  write "Can't transfer";
```

```
|   $88:  write "Null file";
|   $89:  write "Command Table overflow";
|   $8A:  write "Execution Table overflow";
|   $8B:  write "SYSTEM blocks have changed";
|   $8C:  write "Invalid initialisation";
|   else: reporterror(ErrNo);
| end;
| case ErrNo of
| begin
|   $85:$89:$8A:$8B:$8C: EscapeToEOL; ProgramID:=$0000; STOP;
|   else: write "^M^JContinue (Y/N)? "; read Answer;
|         if Answer="N" OR Answer="n" OR Answer=CHR($0D) then
    |          (EscapeToEOL; ProgramID:=$0000; STOP);
    | end;
    end;
  end;

end.

% ----------------------------------------------------------------------



*TITLE="Parse Command Line" (File E4PARSE) 1 MAY 84
% Adapted from module by WHIMSICAL DEVELOPMENTS

module Parse=
begin
public
  BOOL       Catalog,      % True if User Command Table Catalog requested
             ExtraInfo,    % True if block usage info requested
             UnhookLocal;  % True if Local to be unhooked
  SMALLINT   ParseType;    %  1 - File specification
                           %  2 - Options
                           %  3 - End of Command
                           % 10 - Command Line Error
  CHAR ARRAY Name[14];
  proc EscapeToEOL;        % Escape to End of Line
  proc Parse;              % Parses Command Line; Returns Name and ParseType
external
  proc Error(BYTE ErrNo);
private
  CHAR Ch($CC18),          % One character look ahead
       Alpha="A", Num="0",
       Sep=" ",   EOL=CHR($0D);

  proc NextCh=external($C927);


  CHAR proc Class=
% ---- ---- -----
  begin
    CHAR FlexEOL($CC02);
    if Ch>="A" AND Ch<="Z" OR Ch>="a" AND Ch<="z" then Class:=Alpha else
    if Ch>="0" AND Ch<="9" then Class:=Num else
    if Ch=" " OR Ch="," then Class:=Sep else
    if Ch=EOL OR Ch=FlexEOL then Class:=EOL else
    Class:=Ch;
  end;

  proc EscapeToEOL=
% ---- -----------
  begin
    while Class<>EOL do NextCh; % Escape to EOL
  end;

  proc Parse=
% ---- -----
  begin
    BOOL DriveFound;

    CHAR proc FilterCh=
  % ---- ---- --------
    begin
      BYTE ULCFlag($CC49);     % Upper/Lower Case Flag
      if ULCFlag=$60 then      % Map lower case to upper
      begin
        if Class=Alpha then Ch:=CHR(ASC(Ch) AND $5F);
      end;
  end;

  BOOL proc NotEnd=          % True if not separator and not end of line
% ---- ---- ------
  begin
    NotEnd:=Class<>Sep AND Class<>EOL;
  end;

  BOOL proc NotFileCh=       % True if Ch is not a valid filename character
% ---- ---- ---------
  begin
    NotFileCh:=Class<>Alpha AND Class<>Num AND Ch<>"-" AND Ch<>"_";
  end;

  proc ParseDrive=
% ---- ----------
  begin
    if DriveFound then Error($82);
    DriveFound:=true;
    if Ch>="0" AND Ch<="3" then Name[0]:=Ch else Error($80);
    NextCh;
  end;

  proc ParseName=
% ---- ---------
  begin
    BYTE      SysDrive($CC0B);
    SMALLINT NameNdx:=2, ExtNdx;
    Name[0]:=CHR(SysDrive+$30);
    Name[1]:=".";
    if Class=Num then
    begin
      ParseDrive;
      if Ch<>"." then Error($81);
      NextCh;
      if Class<>Alpha then Error($81);
    end;
    FilterCh;
    do begin
      Name[NameNdx]:=Ch;
      NextCh; FilterCh;
      NameNdx:=NameNdx+1;
    end until NotFileCh OR NameNdx=10;
    Name[NameNdx]:=  "."; NameNdx:=NameNdx+1;
    Name[NameNdx]:=  "C"; % Set default extension
    Name[NameNdx+1]:="M";
    Name[NameNdx+2]:="D";
    Name[NameNdx+3]:=" ";
    if Ch="." then
    begin
    : NextCh;
    : case Class of
    : begin
    :   Num:   ParseDrive;
    :   Alpha: FilterCh;
    :          do begin
    :            Name[NameNdx+ExtNdx]:=Ch;
    :            NextCh; FilterCh;
    :            ExtNdx:=ExtNdx+1;
    :          end until NotFileCh OR ExtNdx=3;
    :          Name[NameNdx+ExtNdx]:=" ";
    :          if Ch="." then (NextCh; ParseDrive);
    :   else:  Error($81);

    : end;
    end;
  end;

  proc ParseOpts=
% ---- ---------
  begin
    NextCh;     % Skip "+" to get first option
    while NotEnd do
    begin
    : case Ch of
    : begin
    :   "C":"c": Catalog:=true;
    :   "I":"i": ExtraInfo:=true;
    :   "U":"u": UnhookLocal:=true;
    :   else:    Error($84);
    : end;
    : NextCh;
    end;
  end;

  while Class=Sep do NextCh;   % Skip leading separators
  case Class of
  begin
    Alpha:Num: ParseType:=1; ParseName;
    "+":       ParseType:=2; ParseOpts;
    EOL:       ParseType:=3;
    else:      ParseType:=10;
  end;
end;

end.

% ----------------------------------------------------------------------



"TITLE="Generate a table of 4K Blocks of RAM" (File E4BLOCK) 30 APR 84

module Block=
begin
public
  INTEGER    TotalUsed,
             FreeLeft;
  BYTE       CurrentBlock($EFF0);
  DBYTE      BlockAddress($EFF2);
  BYTE ARRAY Block($0000);
  BYTE proc NextBlock;
  proc RestoreBlocks;
external
  BOOL Err;
  DBYTE ProgramID($EFFE);
  proc Error(BYTE ErrNo);
private
  BYTE       Ndx;                % General purpose index
  DBYTE      Sample($00F0);      % Location where RAM sampled for existance
  BYTE ARRAY Original[$F],       % Original configuration
             Image($DFD0),       % DAT image
             DAT($FFF0),         % DAT
             BlockInfo($EE00);   % Array of block information,
                                 %   BIT 7  allocated to CONTROL
                                 %       6  allocated to SYSTEM
                                 %       5  allocated to LOCAL
                                 %       4  allocated to VDISK
                                 %       3
                                 %       2
                                 %       1
                                 %       0  RAM present

  BOOL proc OriginalBlocks=
% ---- ---- --------------
  begin
    BYTE Ndx;
    for Ndx:=$0 to $F do
    begin
      Original[Ndx]:=Image[Ndx];     % Load original with standard DAT image
      if BlockInfo[Original[Ndx]]<>$41 then OriginalBlocks:=true;
      BlockInfo[Original[Ndx]]:=$41; % Allocate to SYSTEM
    end;
  end;

  proc RestoreBlocks=
% ---- -------------
  begin
    BYTE Ndx;
    for Ndx:=$0 TO $F do
    begin
      Image[Ndx]:=Original[Ndx];
      DAT[Ndx]:=Original[Ndx];
    end;
  end;

  BYTE proc NextBlock=
% ---- ---- ---------
  begin
    BYTE Ndx;
    while BlockInfo[Ndx]<>$01 do
    begin
      Ndx:=Ndx+$01;
      if Ndx=$00 then Error($85);
    end;
    if NOT Err then
    begin
      NextBlock:=Ndx;
      BlockInfo[Ndx]:=$21;
      Image[0]:=Ndx;
      DAT[0]:=Ndx;
      TotalUsed:=TotalUsed+1; FreeLeft:=FreeLeft-1;
    end;
  end;

  proc FindBlocks=
% ---- ----------
  begin
    BYTE Block;
    for Block:=$00 to $FF do
    if BlockInfo[Block]=$00 then
    begin
      Image[0]:=Block;
      DAT[0]:=Block;
      Sample:=COMBINE($98,Block);
      if Sample=COMBINE($99,Block) then BlockInfo[Block]:=$01;
    end;
    RestoreBlocks;              % Restore original configuration
  end;

  if ProgramID<>HEX(1984) then
  begin
    for Ndx:=$00 to $FF do BlockInfo[Ndx]:=$00;
    OriginalBlocks;
    FindBlocks;
    CurrentBlock:=NextBlock;
    BlockAddress:=$0000;
    ProgramID:=HEX(1984);
  end else
  begin
    if OriginalBlocks then Error($8B);
    if BlockInfo[CurrentBlock]<>$21 then Error($8C);
    Image[0]:=CurrentBlock;
    DAT[0]:=CurrentBlock;
  end;
  FreeLeft:=0;
  for Ndx:=$00 to $FF do if BlockInfo[Ndx]=$01 then FreeLeft:=FreeLeft+1;
end.

% ----------------------------------------------------------------



*TITLE='Build Execution Table' (FILE E4EXTBL) 1 MAY 84

module ExTbl=
begin
public
  DBYTE          ExecAddr,
                 XferAddr;
  INTEGER        Index;
  BYTE ARRAY     CurrBlock[127];
  DBYTE ARRAY    BlockAddr[127],
                 LoadAddr[127];
  INTEGER ARRAY Count[127];

  proc Emit(DBYTE REF Addr; BYTE Data);
  proc Emit2(DBYTE REF Addr; DBYTE Data);
  proc ExecTable;
external
  BOOL        Err;
  DBYTE       ExecTableStart,
              ExecTableEnd;
  BYTE        CurrentBlock($EFF0);
  DBYTE       BlockAddress($EFF2);
  BYTE ARRAY Block($0000);
  CHAR ARRAY Name[14];

  proc Error(BYTE ErrNo);
private
  BYTE ARRAY Memory($0000);
```

```
  proc Emit(DBYTE REF Addr; BYTE Data)=
% ---- ----
  begin
    Memory[Addr]:=Data;
    Addr:=Addr+$0001;
  end;

  proc Emit2(DBYTE REF Addr; DBYTE Data)=
% ---- -----
  begin
    Emit(Addr, HIBYTE(Data));
    Emit(Addr, LOBYTE(Data));
  end;

  proc TableEnd=
% ---- --------
  begin
    while Memory[ExecAddr]=        $BD AND
          Memory[ExecAddr+$0001]=$EF AND
          Memory[ExecAddr+$0002]=$00 DO
    begin
      ExecAddr:=ExecAddr+$0008; % Skip constants etc
      if Memory[ExecAddr]=$7E then ExecAddr:=ExecAddr+$0003; % skip xfer addr
    end;
  end;

  proc ExecTable=
% ---- ---------
  begin
    BYTE     MapBlock;
    INTEGER Ndx;

    if NOT Err then
    begin
      while Ndx<=Index do
      begin
        MapBlock:=$A;
        if LoadAddr[Ndx]>$9000 then MapBlock:=$0;
        Emit(ExecAddr, $BD);
        Emit2(ExecAddr, $EF00);
        Emit(ExecAddr, MapBlock);
        Emit(ExecAddr, CurrBlock[Ndx]);
        Emit2(ExecAddr, BlockAddr[Ndx]);
        Emit2(ExecAddr, LoadAddr[Ndx]);
        Emit2(ExecAddr, HEX(Count[Ndx]));
        Ndx:=Ndx+1;
      end;
      Emit(ExecAddr, $7E);
      Emit2(ExecAddr, XferAddr);
      CurrBlock[0]:=CurrBlock[Index];
      BlockAddr[0]:=BlockAddr[Index]+HEX(Count[Index]);
      CurrentBlock:=CurrBlock[0];
      BlockAddress:=BlockAddr[0];
      Count[0]:=0;
      Index:=0;
      if ExecAddr>=ExecTableEnd then Error($8A);
    end;
  end;

  ExecAddr:=ExecTableStart;
  CurrBlock[0]:=CurrentBlock;
  BlockAddr[0]:=BlockAddress;
  TableEnd;
end.

% ----------------------------------------------------------------

*TITLE='File Loader' (File E4LOAD) 25 APR 84

module Load=
begin
public
  LARGEINT TotalBytes;           % Total bytes emitted
  proc LoadFile;
external
  BOOL Err;                      % Error flag
  DBYTE XferAddr;                % Transfer address
  INTEGER Index;                 % Load info index

  CHAR ARRAY    Name[14];        % File name
  BYTE ARRAY    Block[$0000],   % Data loaded into block
                CurrBlock[127]; % Current block
  DBYTE ARRAY   BlockAddr[127], % Block address
                LoadAddr[127];  % Load address
  INTEGER ARRAY Count[127];     % Byte count


  proc Error(BYTE ErrNo);       % Error handler
  BYTE PROC NextBlock;          % Maps in next free block
private

  proc LoadFile=
% ---- --------
  begin
    BYTE Header, Cnt, Data;
    BOOL NotFirstRecord, IterAddrFound;
    DBYTE LAddr, BAddr:=BlockAddr[0];
    BYTE FILE BinFile;

    BOOL proc BinHeader=
 % ---- ---- ---------
    begin
      BinHeader:=(Header=$02 OR Header=$16 OR Header=$00);
    end;

    % Index is 0, CurrBlock[0] is current block,
    % BlockAddr[0] is next free byte within current block,
    % Count[0] is 0.

    trap to Error from open BinFile as Name;
    if NOT Err then trap to Error from
    begin
    | read from BinFile Header;
    | if NOT BinHeader then Error($86);
    | while BinHeader AND NOT EOF(BinFile) AND NOT Err do
    | begin
    |   if Header=$16 then

    |   begin
    |     read from BinFile IterAddr;
    |     IterAddrFound:=true;
    |   end else
    |   if Header=$02 then
    |   begin
    |   | read from BinFile LoadAddr[Index+1], Cnt;
    |   | if NotFirstRecord AND Count[Index]<>0 then
    |   | begin
    |   |   if LAddr<>LoadAddr[Index+1] then
    |   |   begin
    |   |     Index:=Index+1;
    |   |     CurrBlock[Index]:=CurrBlock[Index-1];
    |   |     BlockAddr[Index]:=BAddr;
    |   |     LAddr:=LoadAddr[Index];
    |   |     Count[Index]:=0;
    |   |   end;
    |   | end else
    |   | begin
    |   |   LAddr:=LoadAddr[Index+1];
    |   |   LoadAddr[Index]:=LAddr;
    |   |   NotFirstRecord:=true;
    |   | end;
    |   | TotalBytes:=TotalBytes+EXTEND(DEC(COMBINE($00,Cnt)));
    |   | while Cnt>$00 AND NOT Err do
    |   | begin
    |   |   read from BinFile Data; Cnt:=Cnt-$01;
    |   |   Block[BAddr]:=Data;
    |   |   Count[Index]:=Count[Index]+1;
    |   |   BAddr:=BAddr+$0001; LAddr:=LAddr+$0001;
    |   |   if BAddr=$1000 then
    |   |   begin
    |   |     Index:=Index+1;
    |   |     CurrBlock[Index]:=NextBlock;
    |   |     BAddr:=$0000;
    |   |     BlockAddr[Index]:=BAddr;
    |   |     LoadAddr[Index]:=LAddr;
    |   |     Count[Index]:=0;
    |   |   end;
    |   | end;
    |   end;
    |   if NOT EOF(BinFile) then read from BinFile Header;
    | end;
    | close BinFile;
    | if NOT IterAddrFound then Error($87);
    | if Count[Index]=0 then
    | begin
    |   if Index=0 then Error($88) else Index:=Index-1;
    | end;
    end;
  end;

end.

% ------------------------------------------------------------


*TITLE="Build Command Table" (FILE E4CMDTBL) 30 APR 84

module CmdTbl=
begin
public
  proc CatCmdTable;
  proc CmdTable;
  proc CloseCmdTable;
external
  BOOL        Err;
  DBYTE       CmdTableStart,
              CmdTableEnd,
              ExecAddr;
  CHAR ARRAY Name[14];
  proc Error(BYTE ErrNo);
  proc Emit(DBYTE REF Addr; BYTE Data);
  proc Emit2(DBYTE REF Addr; DBYTE Data);
private
  DBYTE LocTable:=CmdTableStart,
        UserCmdTable($CC12);

  CHAR ARRAY Memory[$0000];

  proc TableEnd=
% ---- --------
  begin
    if UserCmdTable=$0000 then
    begin
      UserCmdTable:=LocTable;
    end else
    begin
      if CmdTableStart<>UserCmdTable then CmdTableEnd:=$FFFF; % UCT not LOCAL's
      CmdTableStart:=UserCmdTable;
      LocTable:=CmdTableStart;
      while Memory[LocTable]<>CHR($00) do
      begin
      | while Memory[LocTable]<>CHR($00) do % Skip name
      | begin
      |   LocTable:=LocTable+$0001;
      | end;
      | LocTable:=LocTable+$0003; % Skip address
      end;
    end;
  end;

  proc CatCmdTable=
% ---- -----------
  begin
    DBYTE Ndx:=CmdTableStart;
    write "^M^JFILES IN USER COMMAND TABLE^M^J";
    while Ndx<LocTable do
    begin
    | write "^M^J";
    | while Memory[Ndx]<>CHR($00) do % write name
    | begin
    |   write Memory[Ndx];
    |   Ndx:=Ndx+$0001;
    | end;
    | Ndx:=Ndx+$0003; % Skip address

     end;
   end;

   proc CmdTable=
 % ---- --------
   begin
     BYTE Ndx:=$2;
    if NOT Err then
    begin
      while Name[Ndx]<>'.' do
      begin
        Emit(LocTable, ASC(Name[Ndx]));
        Ndx:=Ndx+$01;
      end;
      Emit(LocTable, $00);
      Emit2(LocTable, ExecAddr);
      if LocTable>CmdTableEnd then Error($89);
    end;
  end;

  proc CloseCmdTable=
* ---- ------------
  begin
    if UserCmdTable<>$0000 then Emit(LocTable, $00);
  end;

  TableEnd;
end.

* ----------------------------------------------------------------------



* Procedure to Copy a Program Into Main Memory (File E&COPY.ASM) 1 MAY 84
* --------- -- ---- - ------- ---- ---- ------

* by M G Armstrong

          ORG $EF00          Use CPU Board memory

* Execution Tables should be set up as follows:
* JSR EF 00
* FCB MapBlockNdx            Index of block used to map in CurrBlock
* FCB CurrBlock              Block containing the program
* FDB BlockAddr              Start address of program within CurrBlock
* FDB LoadAddr               Start load address of program in main memory
* FDB Count                  The number of bytes to load
* .
* .                          Repeat above sequence as required
* .
* JMP XferAddr               Jump to transfer address

Image     EQU $DFD0          Base address of DAT image
DAT       EQU $FFF0          Base address of DAT
COPY      PULS Y             Pull return address into (Y)
          LEAX 8,Y           Skip the constants
          PSHS X             Push new return address
          LDX #Image         Point (X) to DAT image
          LDA 0,Y            Load MapBlockNdx into (A)
          LDB A,X            Load Image[MapBlockNdx] into (B)
          PSHS D             Save index and original block on stack
          LDB 1,Y            Get CurrBlock
          STB A,X            Image[MapBlockNdx]:=CurrBlock
          LDX #DAT           DAT[MapBlockNdx]:=CurrBlock
          STB A,X            CurrBlock now resident in MapBlock
          ASLA               Calculate base address of CurrBlock
          ASLA
          ASLA
          ASLA
          CLRB
          ADDD 2,Y           AbsoluteBlockAddr:=BASE(CurrBlock)+BlockAddr
          TFR D,U            Transfer AbsoluteBlockAddr to (U)
          LDX 4,Y            Get LoadAddr
          LDY 6,Y            Get Count
COPY1     LDB 0,U+           Do Load Data
          STB 0,X+           Store Data
          LEAY -1,Y          Decrement Count
          BNE COPY1          Until Count=0;
          LDX #Image         Point (X) to DAT image
          PULS D             Recover index and original block
          STB A,X            Image[MapBlockNdx]:=OriginalBlock
          LDX #DAT           DAT[MapBlockNdx]:=OriginalBlock
          STB A,X
          RTS                Return

          END

* ======================================================================
```

FROM TSC BASIC TO MICROSOFT BASIC

by E. M. (Bud) Pass, Ph.D.
Computer Systems Consultants, inc.
1454 Latta Lane, Conyers, GA 30207
Telephone Number 404-483-1717/4570

INTRODUCTION

The TSC BASIC interpreters for FLEX and UNIFLEX are generally excellent implementations of BASIC for the earlier 6800 systems under FLEX and current 6809 systems under FLEX and UNIFLEX. A large amount of business, educational, recreational, technical and system software has been developed using those interpreters. With the advent of the Radio Shack TRS-80 Color Computer, the IBM Personal Computer, and other newer systems supporting Microsoft and similar BASIC implementations, many developers have been and are converting these TSC BASIC programs to Microsoft BASIC implementations.

The purpose of this article is to discuss the differences between TSC BASIC and Microsoft BASIC, and how they affect the conversion from the former to certain implementations of the latter.

MICROSOFT BASIC

Microsoft BASIC implementations have been performed on a large number of systems, unlike TSC BASIC, which has been implemented on a rather limited number. Microsoft offers both BASIC interpreters and compilers, with the advantages of size, speed, and security of a compiler, while still maintaining the ease of development of an interpreter. TSC offers only interpreters; their Pre-Compiler is not a compiler but rather is a tokenized interpreter.

Unfortunately, not all of Microsoft's (or TSC's) implementations are the same, so some attention must be given to the differences among the implementations of interest. The implementations discussed here are for the Color Computer and for the IBM Personal Computer, being fairly recent and complete implementations, with extensions in similar areas for supporting color graphics and sound effects. Other implementations are generally similar in the core language, but differ in the extensions. Developers aware of the similarities and differences among the implementations can use the similarities to their advantage while minimizing the impact of the differences.

DIFFERENCES AND SIMILARITIES

This section provides a discussion of the primary points of difference and similarity between the TSC BASICs and the Microsoft BASICs. Note that the graphics, sound, and certain and other extended functions, such as cassette I/O, of some of the major Microsoft BASICs are not covered here since the thrust of this discussion is the translation of TSC BASIC programs to Microsoft BASIC, and TSC BASIC does not support graphics, sound, or the extended functions.

For the purpose of this discussion, the following mnemonics will be used as shorthand notation for the indicated implementations:

TSC	any TSC Extended BASIC (here)
XBASIC	TSC Extended BASIC Interpreter
XPC	TSC Extended BASIC Pre-Compiler
Microsoft	any Microsoft BASIC (here)
PC	IBM PC BASIC
COLOR	Radio Shack Extended BASIC

Naming Conventions

XBASIC supports a one or two character variable name, starting with a letter, and optionally followed by a letter or digit. It requires every statement to be labelled with a numeric integer in the range from 1 to 32767.

XPC supports a variable name of length one to 255 characters, starting with a letter, and composed of letters, digits, and underlines. It does not require every statement to be labelled, allowing only those statements which are the targets of GOTO, GOSUB, ERL, etc. to be labelled, and allows labels to be numeric integers or to follow the same rules as do variable names; a label must start in the first column.

TSC allows a variable name to be followed by '$' or '%', denoting that the variable represents a string or integer, respectively, rather than a floating point number. The optional suffix character is considered a part of the name.

COLOR supports a variable name of effectively any length, starting with a letter, and optionally followed by a letters and digits; however, only the first two characters are significant. It requires every statement to be labelled with a numeric integer in the range from 1 to 32767. It allows a variable name to be followed by '$', denoting that the variable represents a string, rather than a floating point number. The optional suffix character is considered a part of the name, although it is exempt from the two-character rule.

PC supports a variable name of effectively any length, starting with a letter, and optionally followed by a letters, digits, and periods; however, only the first 40 characters are significant. It requires every statement to be labelled with a numeric integer in the range from 0 to 65529. PC allows a variable name to be followed by '$', '%', '!', or '#', explicitly denoting that the variable represents a string, integer, short floating point number, or long floating point number, respectively, rather than either a short floating point number or the default declaration type implied by a DEFtype statement. The optional suffix character is considered a part of the name, although it is exempt from the 40 character rule. It also allows a numeric constant to be followed by '!' or contain the exponential form 'E' to force its representation as a short floating point number, or to be followed by '#' or contain the exponential form 'D' to force its representation as a long floating point number.

None of the BASICs discussed here distinguish between upper and lower case in variable, verb, or function names.

All of the BASICs discussed here allow variables to be subscripted, with one or two dimensions, through the use of the DIM statement. TSC extends this concept with virtual arrays, which are actually random disk files, rather than tables in memory. All of them automatically clear numeric variables to zero and string variables to null. This includes subscripted variables, but not virtual arrays.

The diversity of legal names for XPC may cause compatibility problems when converting to Microsoft because of the possibility of using names which are proper in XPC, but are reserved words in the target Microsoft implementation. Usually, this will produce syntax errors, but sometimes will cause other problems, such as accidentally changing the system date or time, as will storing into DATE$ or TIME$ on PC. A good defense against this problem involves reviewing a sorted cross reference listing of the XPC program versus a list of the reserved words for the target language, and modifying the offending variable names.

Another problem caused by the differences among the naming conventions concerns the possibility that two unique XPC variables may be interpreted ambiguously as one variable in either COLOR or PC. While it is unlikely that two XPC variables would be the same for the first 40 characters, but different thereafter, causing a problem under PC, it is quite possible that two XPC variables would be the same for the first two characters, but different thereafter, causing a problem under COLOR. Again, the best defense involves reviewing a sorted cross reference listing of the XPC program for ambiguous names.

String and Numeric Representation

TSC supports strings of length zero to 32767 bytes and the following numeric representations:

- integer
  - -32768 to +32767
  - 2 bytes
- floating point
  - 17 digits
  - 8 bytes

COLOR supports strings of length zero to 255 bytes and the following numeric representations:

- integer
  - -32768 to +32767
  - 2 bytes
- floating point
  - 7 digits
  - 5 bytes

Note that the number of digits of precision provided by COLOR is only seven, and this may be insufficient for the purposes of the program. For instance, accounting programs on COLOR will be unable to exactly compute amounts greater than 99999.99 in magnitude using the floating point arithmetic provided, assuming two decimal places are required for dollars and cents representation.

PC supports strings of length zero to 255 bytes and the following numeric representations:

- integer
  - -32768 to +32767
  - 2 bytes
- short floating point
  - 6 digits
  - 4 bytes
- long floating point
  - 17 digits
  - 8 bytes

Since the PC default for short floating point provides only six digits of precision, compounding the accuracy problem discussed for COLOR, the following statement should normally be inserted in each program being converted from TSC to PC:

```
    DEFDBL A-Z
```

before any other statements or declarations to cause the default declaration of PC variables to be long floating point, and thus avoid any loss of precision. Generally, the benefits derived from the use of this statement outweigh its cost; however, programs which are time or space critical should be more carefully reviewed to determine if the use short floating point may be more appropriate for some or all of the floating point arithmetic. PC has one other peculiarity not common among other BASICs in that it rounds, rather than truncates, when converting floating point numbers to integer format; this may cause subtle problems in many programs.

All the BASICs discussed here allow arbitrary contents for strings (as opposed to DG BASIC, which uses hex 00 to flag end of strings, etc.), although they all have length limits. The length limitation of Microsoft strings to 255 characters will cause no problems in some TSC programs and severe problems in other TSC programs being converted. A general solution to the problem is not possible. Program logic must generally be carefully reviewed while testing to ensure that the 255 character length limitation is resolved. Usually, the BASIC interpreter or compiler will detect and flag such problems; however, the problems may be masked (at least under PC) by error handling routines not expecting string length errors.

Microsoft allows hexadecimal and octal constants to be explicitly coded, as follows:

```
   hexadecimal
         &Hxxxx     (x=0-9,A-F)
   octal
         &Oxxxxxx  (x=0-7)
         &xxxxxx
```

Hexadecimal and octal constants may be used in the same contexts as integer constants. In some Microsoft programs, the use of hexadecimal constants may alleviate some or all of the problems caused by the lack of the TSC HEX string conversion function, discussed later.

Operators and Expressions

All the BASICs discussed here share a similar set of arithmetic, string, and logical operators. PC has several unique operators ('\', MOO, XOR, EQV, IMP). UNIFLEX TSC has one unique operator (' '), and Microsoft has several ('><', '=>', '=<') not supported by TSC.

The table below provides a composite list of all of the BASIC operators, in decreasing hierarchial order:

(,)	parentheses
fcn()	functions
¦	exponenation (caret)
-,+	unary negative/positive
*,/	multiplication/division
\,MOO	integer division/remainder
+,-	addition/subtraction/ string concatenation
<>,><,<=,=<,>=,=>,<,>,=,	relational comparisons
NOT	logical complement
AND	logical conjunction
OR	logical disjunction
XOR	logical excl. disjunction
EQV	logical equivalence
IMP	logical implication

PC interprets the division operator ('/') as always producing a floating point result, whereas the other BASICs interpret it as producing a truncated integer result when both operands are integers. This is consistent with the fact that PC rounds, rather than truncating, when converting floating point numbers to integer format. The PC operator ('\') converts both of its operands to integers and produces a truncated integer result. The Uniflex approximately equal operator (' ') may usually be converted to the more conventional equal operator ('='), but each use must be evaluated. All of these differences will require attention in many programs to avoid subtle problems, such as a subscript value being incorrect by one, a relational operation that is never true, etc.

All of the BASICs discussed here allow Boolean expressions to be used in arithmetic contexts, returning non-0 for true and 0 for false. They interpret the logical operators as bitwise, rather than true/false. They all have the same operator hierarchies. They all interpret a binary '+' operator in a string context to represent concatenation.

They all evaluate expressions involving operators of equal precedence on a left to right basis, except for those involving exponentiation, which are evaluated on a right to left basis (for the exponentiation operation only).

Multiple Statements per Line

All of the BASICs discussed here support (and encourage) the placement of multiple statements per line, and all interpret the concatenated statements in a similar manner. TSC allows either ':' or '\' as representing statement concatenation, whereas Microsoft allows only ':'. All of them allow lines of up to 255 characters in length. XPC allows lines to be continued by the use of the '\' and carriage return combination. PC allows lines to be continued by the use of a line feed, rather than a carriage return, although the 255 character limit applies to the entire concatenated statement, even on multiple lines, as opposed to XPC, which imposes the limit only on each line of a multiple line multiple statement. These considerations may occasionally cause problems beyond simple character substitution, but such problems occur rarely in practice.

Non-I/O Functions and Statements

This section summarizes the primary differences among the TSC and Microsoft BASICs in terms of the non-I/O functions and statements. I/O functions and statements will be discussed in the next section.

The string manipulation functions LEFT$, MID$, and RIGHT$ are common in syntax and interpretation across all of the BASICs discussed here. However, many of the other string functions supported by TSC are either not supported by either COLOR or PC or are supported in a different manner.

The CVT group of TSC string conversion functions generally has correspondences under different names under PC and has only partial correspondences under COLOR. CVT$% corresponds to CVI and CVT%$ corresponds to MKI$ in both PC and COLOR, converting a two character internal representation of an integer to and from an integer. CVT$F corresponds to CVD and CVTF$ corresponds to MKD$ in PC only, converting an eight character internal representation of a floating point number to and from a floating point number. CVT$F loosely corresponds to CVN and CVTF$ loosely corresponds to MKN$ in COLOR only, in that the COLOR string functions process a five character internal representation of a floating point number. CVT$F also loosely corresponds to CVF and CVTF$ loosely corresponds to MKF$ in PC only, in that those PC string functions process a four character internal representation of a floating point number.

The TSC STR$ function always provides a trailing space, but the Microsoft STR$ function never provides a trailing space. PC supports a DATE$ function, but it returns a string with format MM-DD-YYYY, not DD-MMM-YY, as returned by the TSC DATE$ function. PC supports a TIME$ function, but it returns a string with format HH:MM:SS, not as returned by TSC UNIFLEX. COLOR does not support either DATE$ nor TIME$. Microsoft does not support the TSC UNIFLEX options of the OATE$ and TIME$ functions with parameters.

The TSC ASC function will accept a null argument, returning a zero, but the Microsoft ASC function requires a non-null argument; a simple manner to avoid this problem is to suffix the arguments of all questionable ASC functions with "+CHR$(0)". Microsoft does not support the TSC UNIFLEX MEM function, but it does support the TSC FLEX FRE function.

Microsoft has no equivalent for the TSC HEX function, which converts its argument from a string containing a hexadecimal number to an integer representing that number. At each occurrence of the use of HEX must be inserted substitute code to perform the function. Microsoft does not allow DEF functions with string arguments; however, a USR function call could potentially be substituted for the HEX function call to perform the conversion.

The error-handling capabilities of TSC and PC are syntactically identical, both using the ON ERROR and RESUME statements and the ERR and ERL functions to establish error-handling routines and to return to normal processing after an error has been detected and processed. The primary differences between the implementations lie in the interpretation of the error numbers returned by the ERR functions and in the fact that, once the PC version of ERR provides an error number, it will return zero until another error occurs. The following list provides a few of the most important error conditions and the values returned by the respective ERR functions:

TSC	PC	Condition
4	53	File Not Found
8	62	End Of File
9	57	I/O Error
16	71	Disk Drive Not Ready
80	7	Out Of Memory

COLOR does not support most error handling, severely limiting its ability to escape from error situations. Microsoft supports the EOF function, which indicates an end of file condition; this is the only error handling function internally provided by COLOR.

Microsoft has no equivalents for the TSC UNIFLEX multitasking statements and functions (such as SLEEP, TASK$, TERM$, TSTAT%, UNLOCK), nor for the TSC statement EXEC. Such functions must be either deleted or replaced by USR functions to request the operating system to perform similar tasks.

Microsoft supports the TSC FLEX PEEK and USR functions under the same names and the PTR function under the name VARPTR. It also supports the POKE statement. However, the manner in which USR and PTR functions are handled is different in essentially every implementation. Also, the uses of the PEEK function and POKE statement are highly dependent upon the hardware and software configuration on which the program is expected to run. Thus every occurrence of any of these functions and statements must be carefully evaluated in every case.

I/O Functions and Statements

This section summarizes the primary differences among the TSC and Microsoft BASICs in terms of the I/O functions and statements.

I/O file numbers are used in similar manners by TSC and Microsoft, although there are several differences in interpretation. One potentially major difference concerns the use of file number zero. TSC interprets file number zero to be the user's terminal, unless the file is opened for output, in which case it is interpreted to be a printer, or unless the file is opened for input, in which case the input prompts to the terminal are deleted. Microsoft does not support any use of file number zero, so that any TSC programs using it will require modification. PC supports only three file numbers (1-3) by default; however, a command line parameter may be used to increase this number to twelve, if enough memory is available to support that many buffers.

Another important area of difference concerns file specifiers. File naming rules are generally more dependent upon operating system requirements than upon BASIC conventions. The BASICs discussed here conform to the following four sets of file naming rules, discussed below:

TSC FLEX
TSC UNIFLEX
COLOR
PC

TSC FLEX file specifiers reference disk files only. They are composed of an optional drive number (0-3), a file name of 1 to 8 characters, and an optional suffix of 1 to 3 characters. The file name and suffix must start with a letter and may be composed of letters, digits, and certain special characters. The drive number, if present, is separated from the file name with a colon. The suffix, if present, is separated from the file name with a period. Letter case is significant. If drive number is omitted, the default work drive is assumed.

TSC UNIFLEX file specifiers reference any device. They are composed of a tree-structured file reference, which is an optional set of directory levels separated by slashes, followed by a file name. Each directory level and file name must start with a letter and may be composed of letters, digits, and certain special characters, excluding slashes. If the file specifier is not preceded by a slash, UNIFLEX prepends pre-specified directory levels to the name. Letter case is significant.

COLOR file specifiers reference disk files only. They are composed of a file name of 1 to 8 characters, an optional suffix of 1 to 3 characters, and an optional drive number (0-3). The file name and suffix must start with a letter and may be composed of letters, digits, and certain special characters. The suffix, if present, is separated from the file name with a slash or period. The drive number, if present, is separated from the file name or suffix with a colon. Letter case is insignificant. If drive number is omitted, the default drive number is assumed.

PC file specifiers reference any device. They are composed of an optional device id, an optional file name of 1 to 8 characters, and an optional suffix of 1 to 3 characters. The file name and suffix, if present, must start with a letter and may be composed of letters, digits, and certain special characters. The device id, if present, is separated from the file name with a colon. The suffix, if present, is separated from the file name with a period. Letter case is insignificant. PC device ids are interpreted as follows:

A:	disk drive A
B:	disk drive B
C:	disk drive C
D:	disk drive D
CAS1:	cassette adapter
COM1:	communications adapter 1
COM2:	communications adapter 2
KYBD:	keyboard adapter
LPT1:	printer adapter 1
LPT2:	printer adapter 2
LPT3:	printer adapter 3
SCRN:	screen adapter

If the device id is omitted, the currently assigned disk drive is used. File names are required for disk and are optional for all other device types.

The OPEN statements perform essentially the same functions in all of the BASICs discussed here; however, the formats for the statements are different. All require a file number, a file specifier, a mode, and some allow a logical record length to be stated. As just noted, TSC allows file number zero, whereas Microsoft does not. Also, the formats for file specifiers differ among the versions of BASIC, as do the interpretations of the modes.

TSC supports the following formats for OPEN statements:

```
    OPEN OLD filespec AS filenumb
    OPEN NEW filespec AS filenumb
    OPEN     filespec AS filenumb
```

and TSC UNIFLEX supports the following additional parameter for random files only:

```
    ,SIZE recordsize
```

where recordsize specifies the length of all records in a random file; by default, it is assumed to be 252, which is the same record length always used by TSC FLEX for random files. Mode OLD requires the disk file to pre-exist and opens the file for input only. Mode NEW always creates a new file, deleting any old one by the same name on the same drive, and opens the file for output only. The null mode opens an existing file or creates a new one, and opens the file for both input and output (random access only).

COLOR supports the following formats for OPEN statements:

```
    OPEN mode, filenumb, filespec
    OPEN mode, #filenumb, filespec
```

where mode is a string expression with the following interpretations of the first character in the string:

I	input
O	output
R	random

and COLOR supports the following additional parameter for random files only:

```
    ,recordsize
```

where recordsize specifies the length of all records in a random file; by default, it is assumed to be 256. For conversion from TSC FLEX, the recordsize parameter should be stated as 252. For conversion from TSC UNIFLEX, it should be stated as the same value stated or assumed originally, unless the value is greater than 256, in case further manual intervention will be required to reduce the record length or split up length always used by TSC FLEX for random files. Mode "I" requires the disk file to pre-exist and opens the file for input only. Mode "O" always creates a new file, deleting any old one by the same name on the same drive, and opens the file for output only. Mode "R" opens an existing file or creates a new one, and opens the file for both input and output (random access only).

PC supports the following formats for OPEN statements:

```
    OPEN filespec FOR APPEND AS filenumb
    OPEN filespec FOR APPEND AS #filenumb
    OPEN filespec FOR INPUT  AS filenumb
    OPEN filespec FOR INPUT  AS #filenumb
    OPEN filespec FOR OUTPUT AS filenumb
    OPEN filespec FOR OUTPUT AS #filenumb
    OPEN filespec            AS filenumb
    OPEN filespec            AS #filenumb
    OPEN mode, filenumb, filespec
    OPEN mode, #filenumb, filespec
```

where mode is a string expression with the following interpretations of the first character in the string:

I	input
O	output
R	random

and PC supports the following additional parameters for random files only:

```
    ,LEN=recordsize   (for OPEN filespec)
    ,recordsize       (for OPEN mode)
```

where recordsize specifies the length of all records in a random file; by default, it is assumed to be 128. For conversion from TSC FLEX, the recordsize parameter should be stated as 252. For conversion from TSC UNIFLEX, it should be stated as the same value stated or assumed originally, unless the value is greater than 1024, in case further manual intervention will be required to reduce the record length or length always used by TSC FLEX for random files. Mode "I" (or INPUT) requires the disk file to pre-exist and opens the file for input only. Mode "O" (or OUTPUT) always creates a new file, deleting any old one by the same name on the same drive, and opens the file for output only. Mode "R" (or null) opens an existing file or creates a new one, and opens the file for both input and output (random access only). In many cases, it is necessary to attempt to open a random file as an input file, close it, then open it random, to prevent the automatic creation of the random file caused by the PC random file OPEN statement. If a printer is opened as mode "R" and record length 255, the normal automatic line feed after carriage return will be suppressed. Mode APPEND opens an existing file or creates a new one, and opens the file for output only, starting at the end of the file. File records may be no longer than 128 bytes, by default; however, this limit may be increased to 1024 and the default limit of file numbers 1-3 may be increased to 1-12 (both memory size permitting) thru the use of command line parameters.

TSC FLEX sets the width of a printer thru TTYSET parameters, which are external to BASIC, but may be manipulated from BASIC thru the use of the EXEC statement or may be established before BASIC is executed. TSC UNIFLEX sets the width of the printer with the following statement:

```
    WIDTH width
```

PC sets the width of the printer with the following statements:

```
    WIDTH filenumb,width
    WIDTH deviceid,width
```

with which the first type statement requires that the file be open and the second type does not; also, the second form applies to any access to the device, and thus affects the LLIST and LPRINT statements, which the first form does not. In most cases, the second form should be used. However, the first form may be required if several printers, with different widths, must be driven, and, for some reason, the second form is not convenient to use.

In order to overcome the problem caused by the use by TSC of file number zero, the Microsoft LPRINT statement may in many cases be used to replace the TSC PRINT statement. This has the advantage of being a simple substitution, but unconditionally sends its output to the printer, whereas TSC printer output is conditional on an OPEN statement attaching a printer driver to file number zero, and normally always sends its output to a particular printer ("LPT1:" under PC). If this is not convenient, the file number may be changed to a legal one and the file may be opened to device "SCRN:" to send output to the PC screen or "LPT1:", etc. to send output to alternate devices. The other use of the TSC file number zero, to inhibit input prompts, requires manual intervention and review, if it is to be maintained.

There are several other miscellaneous differences among the corresponding I/C statements in TSC and Microsoft. Most of them are minor and will require only cursory review and simple modification. The most important differences are summarized below.

In the MICROSOFT statement "PRINT#n", "n" must be followed by a comma, even if there are no other parameters. Also, in the MICROSOFT "PRINT USING s$" statement, all delimiters after "s$" must be semicolons, whereas TSC allows semicolons or commas.

The only manner in which to set the cursor on the PC screen to a given position is to use the following statement:

```
    LOCATE row,column,cursor,start,stop
```

where row represents the row number (1-25), col represents the column number (1-80), cursor determines whether the cursor is invisible or visible (0,1), start is the cursor start scan line (0-31), and stop represents the cursor stop scan line (0-31). The only manner in which to set the cursor on the COLOR screen to a given position is to use a PRINT statement of the following format:

```
    PRINTn, ...
```

where n represents the character number on the screen, which represents the following expression:

```
    ((row-1)*32)+column)
```

with row values from 1 to 16 and column values from 1 to 32. TSC has no standard for setting the cursor or issuing other terminal commands, but the PRINT statement is normally used to output a character string representing commands to the terminal; this will require review.

The word RECORD in the TSC statements GET and PUT must be deleted when converting those statements to their Microsoft equivalents. Also, the statement "INPUT LINE" must be reversed to "LINE INPUT" when converting a program to Microsoft BASIC.

Although the syntax for the INPUT statements is similar among the implementations, there are differences. The TSC INPUT statement always starts on a new line, whereas the Microsoft INPUT statement does not necessarily start on a new line. If a comma follows the prompt string in a Microsoft INPUT statement, rather than a semicolon, the question mark normally output after the prompt is omitted.

The TSC function "INCH$(n)" inputs one character from file "n". The nearest Microsoft equivalent function is "INPUT$", which has the following forms:

```
    INPUT$(1)
    INPUT$(1,n)
```

where the first form inputs from the keyboard and the second form inputs from file "n". In addition, PC allows the "INKEY$" function, which returns a zero, one, or two character string if no key is struck, a normal key is struck, or an extended key is struck, respectively.

The Microsoft CHAIN statement does not close files which are currently open, unlike the TSC CHAIN statement, which closes all open files. PC supports the "CHAIN MERGE" option, which causes new BASIC text to be read into memory as an overlay, rather than as a replacement.

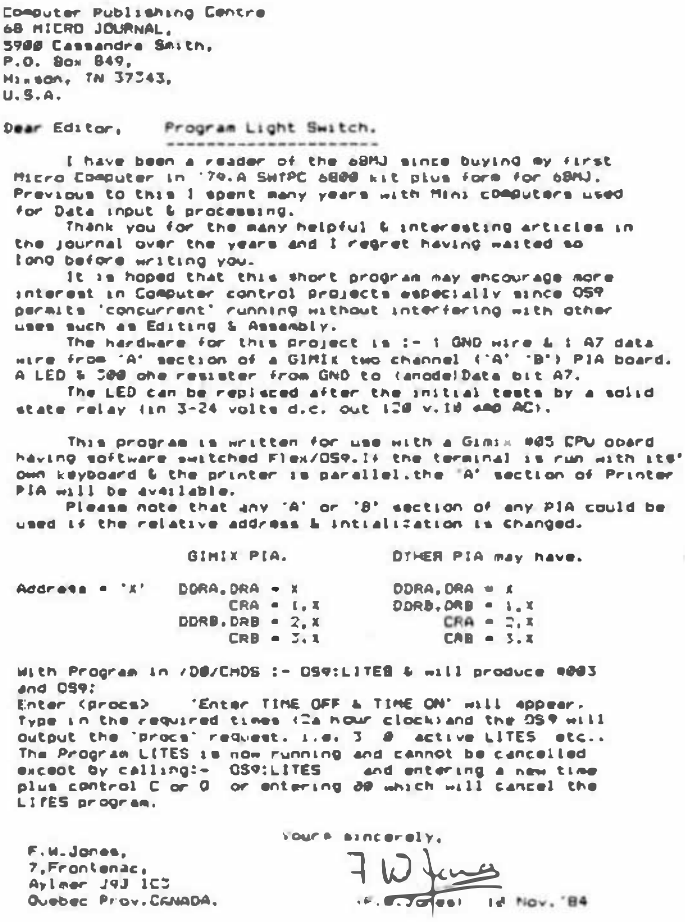
SUMMARY

This article has discussed the similarities and differences among the various implementations of TSC BASIC and Microsoft BASIC from the viewpoint of the conversion of TSC BASIC programs to Microsoft BASIC. In particular, it covered the peculiarities of the COLOR and PC BASIC Microsoft implementations and the TSC FLEX and UNIFLEX, Extended BASIC and Pre-compiler versions and how they affect the conversion process.

The following trademarks were used in this article:

TSC is Technical Systems Consultants, Inc.
FLEX and UNIFLEX are trademarks of TSC.
Radio Shack is a trademark of Tandy, Inc.
TRS-80 Color Computer is a trademark
of Tandy, Inc.
IBM is International Business Machines.
PC is a trademark of IBM.

BIT BUCKET



Computer Publishing Centre
68 MICRO JOURNAL,
5900 Cassandra Smith,
P.O. Box 849,
Hixson, TN 37343,
U.S.A.

Dear Editor,   Program Light Switch.

I have been a reader of the 68MJ since buying my first Micro Computer in '79.A SWTPC 6800 kit plus form for 68MJ. Previous to this I spent many years with Mini computers used for Data input & processing.

Thank you for the many helpful & interesting articles in the journal over the years and I regret having waited so long before writing you.

It is hoped that this short program may encourage more interest in Computer control projects especially since OS9 permits 'concurrent' running without interfering with other uses such as Editing & Assembly.

The hardware for this project is :- 1 GND wire & 1 A7 data wire from 'A' section of a GIMIX two channel ('A' 'B') PIA board. A LED & 300 ohm resister from GND to (anode)Data bit A7.

The LED can be replaced after the initial tests by a solid state relay (in 3-24 volts d.c. out 120 v.10 amp AC).

This program is written for use with a Gimix #05 CPU board having software switched Flex/OS9.If the terminal is run with its' own keyboard & the printer is parallel.the 'A' section of Printer PIA will be available.

Please note that any 'A' or 'B' section of any PIA could be used if the relative address & intialization is changed.

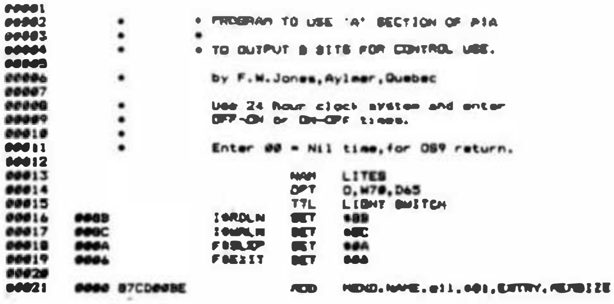
```
                    GIMIX PIA.              OTHER PIA may have.

Address = 'X'    DDRA,DRA = X            DDRA,DRA = X
                      CRA = 1,X          DDRB,DRB = 1,X
                 DDRB,DRB = 2,X               CRA = 2,X
                      CRB = 3,X               CRB = 3,X
```

With Program in /D0/CMDS :- OS9:LITES & will produce #003 and OS9:
Enter (procs)   'Enter TIME OFF & TIME ON' will appear.
Type in the required times (24 hour clock)and the OS9 will output the 'procs' request. i.e. 3 0 active LITES etc..
The Program LITES is now running and cannot be cancelled except by calling:- OS9:LITES   and entering a new time plus control C or Q or entering 00 which will cancel the LITES program.

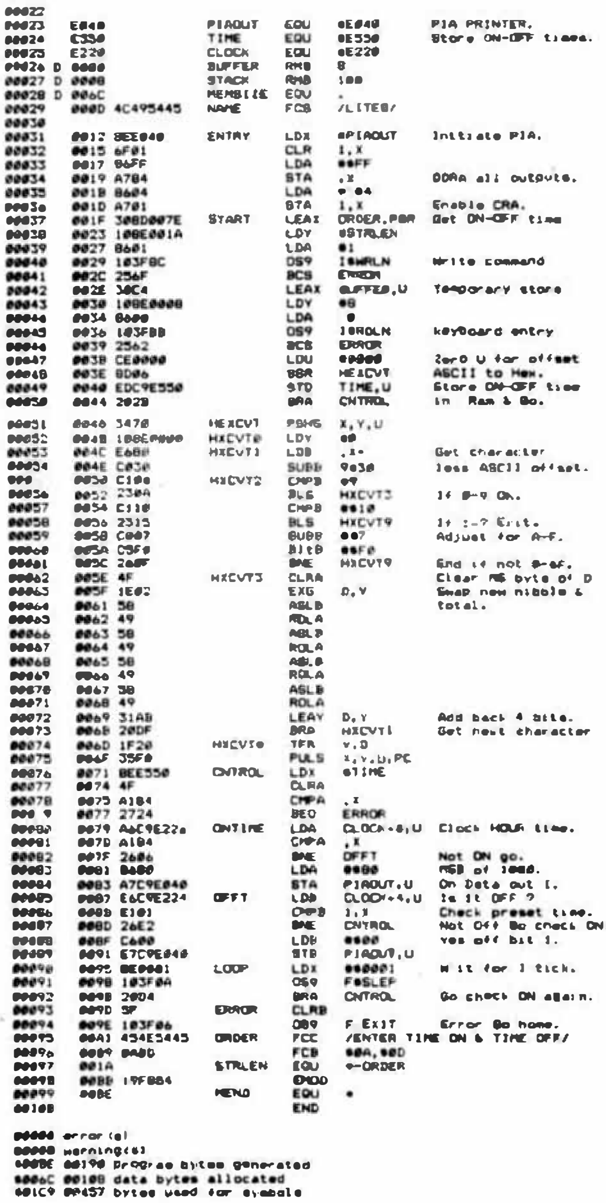
Yours sincerely,

F.W.Jones,
7,Frontenac,
Aylmer J9J 1C3
Quebec Prov.CANADA.

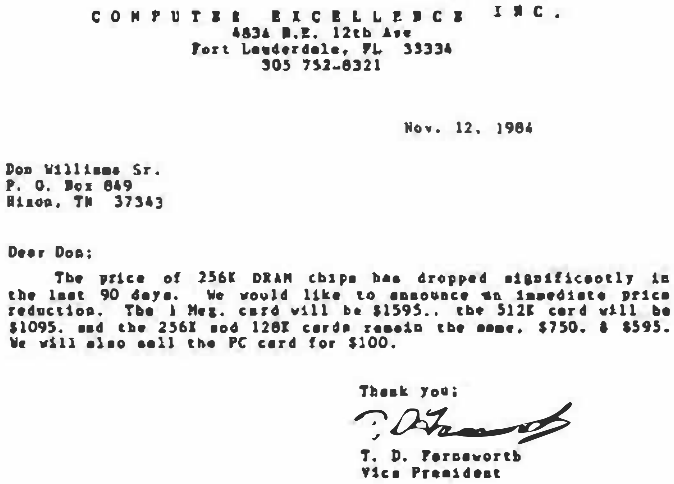
F W Jones
(F.W.Jones)   1st Nov.'84



```
00001
00002            *              * PROGRAM TO USE 'A' SECTION OF PIA
00003            *              *
00004            *              * TO OUTPUT 8 BITS FOR CONTROL USE.
00005
00006            *                by F.W.Jones,Aylmer,Quebec
00007
00008            *                Use 24 hour clock system and enter
00009            *                OFF-ON or ON-OFF times.
00010            *
00011            *                Enter 00 = Nil time,for OS9 return.
00012
00013                                          NAM    LITES
00014                                          OPT    D,W70,D65
00015                                          TTL    LIGHT SWITCH
00016      008B                   I$RDLN       SET    $8B
00017      008C                   I$WRLN       SET    $8C
00018      000A                   F$SLEP       SET    $0A
00019      0006                   F$EXIT       SET    $06
00020
00021      0000 87CD00BE                       MOD    MEND,NAME,$11,$81,ENTRY,MEMSIZE
```



```
      E040              PIAOUT   EQU   $E040       PIA PRINTER.
      E550              TIME     EQU   $E550       Store ON-OFF times.
      E220              CLOCK    EQU   $E220
D     0000              BUFFER   RMB   8
D     0008              STACK    RMB   100
D     006C              MEMSIZE  EQU   *
      000D 4C495445     NAME     FCB   /LITE8/

      0012 8EE040       ENTRY    LDX   #PIAOUT     Initiate PIA.
      0015 6F01                  CLR   1,X
      0017 86FF                  LDA   #$FF
      0019 A784                  STA   ,X          DDRA all outputs.
      001B 8604                  LDA   # 04
      001D A701                  STA   1,X         Enable CRA.
      001F 308D007E     START    LEAX  ORDER,PCR   Get ON-OFF time
      0023 108E001A              LDY   #STRLEN
      0027 8601                  LDA   #1
      0029 103F8C                OS9   I$WRLN      Write command
      002C 256F                  BCS   ERROR
      002E 30C4                  LEAX  BUFFER,U    Temporary store
      0030 108E0008              LDY   #8
      0034 8600                  LDA   0
      0036 103F8B                OS9   I$RDLN      keyboard entry
      0039 2562                  BCS   ERROR
      003B CE0000                LDU   #0000       Zero U for offset
      003E 8D06                  BSR   HEXCVT      ASCII to Hex.
      0040 EDC9E550              STD   TIME,U      Store ON-OFF time
      0044 2028                  BRA   CNTROL      in  Ram & Go.

      0046 3470         HEXCVT   PSHS  X,Y,U
      0048 108E0000     HXCVT0   LDY   #0
      004C E680         HXCVT1   LDB   ,X+         Get character
      004E C030                  SUBB  #$30        less ASCII offset.
      0050 C109         HXCVT2   CMPB  #9
      0052 230A                  BLS   HXCVT3      If 0-9 Ok.
      0054 C110                  CMPB  #$10
      0056 2315                  BLS   HXCVT9      If :-? Exit.
      0058 C007                  SUBB  #07         Adjust for A-F.
      005A C5F0                  BITB  #$F0
      005C 260F                  BNE   HXCVT9      End if not 0-ef.
      005E 4F           HXCVT3   CLRA              Clear MS byte of D
      005F 1E02                  EXG   D,Y         Swap new nibble &
                                                   total.
      0061 58                    ASLB
      0062 49                    ROLA
      0063 58                    ASLB
      0064 49                    ROLA
      0065 58                    ASLB
      0066 49                    ROLA
      0067 58                    ASLB
      0068 49                    ROLA
      0069 31AB                  LEAY  D,Y         Add back 4 bits.
      006B 20DF                  BRA   HXCVT1      Get next character
      006D 1F20         HXCVT9   TFR   Y,D
      006F 35F0                  PULS  X,Y,U,PC
      0071 BEE550       CNTROL   LDX   #TIME
      0074 4F                    CLRA
      0075 A184                  CMPA  ,X
      0077 2724                  BEQ   ERROR
      0079 A6C9E226     ONTIME   LDA   CLOCK+6,U   Clock HOUR time.
      007D A184                  CMPA  ,X
      007F 2606                  BNE   OFFT        Not ON go.
      0081 8680                  LDA   #$80        MSB of 1000.
      0083 A7C9E040              STA   PIAOUT,U    On Data out I.
      0087 E6C9E224     OFFT     LDB   CLOCK+4,U   Is it OFF ?
      008B E101                  CMPB  1,X         Check preset time.
      008D 26E2                  BNE   CNTROL      Not Off Go check ON.
      008F C600                  LDB   #$00        yes off bit 1.
      0091 E7C9E040              STB   PIAOUT,U
      0095 8E0001       LOOP     LDX   #$0001      Wait for 1 tick.
      0098 103F0A                OS9   F$SLEEP
      009B 20D4                  BRA   CNTROL      Go check ON again.
      009D 5F           ERROR    CLRB
      009E 103F06                OS9   F$EXIT      Error Go home.
      00A1 454E5445     ORDER    FCC   /ENTER TIME ON & TIME OFF/
      00B9 0A0D                  FCB   $0A,$0D
      001A              STRLEN   EQU   *-ORDER
      00BB 19F884                EMOD
      00BE              MEND     EQU   *
                                 END

00000 error(s)
00000 warning(s)
$00BE 00190 program bytes generated
$006C 00108 data bytes allocated
$01C9 00457 bytes used for symbols
```



COMPUTER EXCELLENCE INC.
4836 N.E. 12th Ave
Fort Lauderdale, FL 33334
305 752-8321

Nov. 12, 1984

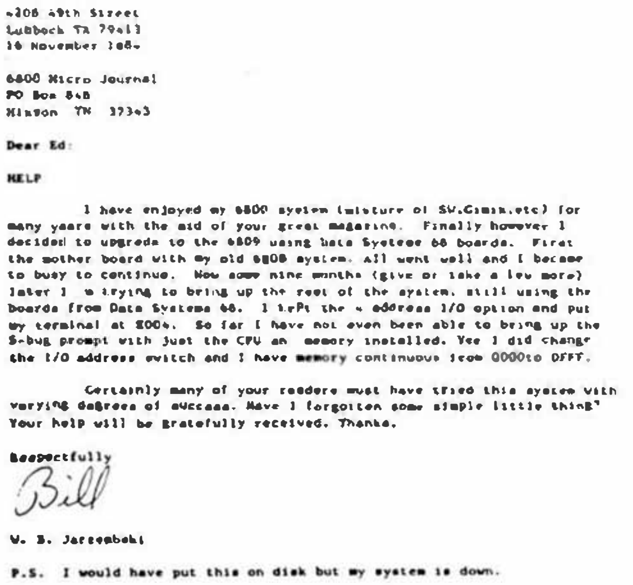
Don Williams Sr.
P. O. Box 849
Hixon, TN 37343

Dear Don;

The price of 256K DRAM chips has dropped significantly in the last 90 days. We would like to announce an immediate price reduction. The 1 Meg. card will be $1595.. the 512K card will be $1095. and the 256K and 128K cards remain the same. $750. & $595. We will also sell the PC card for $100.

Thank you;

T. D. Farnsworth
Vice President



4208 49th Street
Lubbock TX 79413
16 November 1984

6800 Micro Journal
PO Box 849
Hixson TN 37343

Dear Ed:

HELP

I have enjoyed my 6800 system (mixture of SW,Gimix,etc) for many years with the aid of your great magazine. Finally however I decided to upgrade to the 6809 using Data Systems 68 boards. First the mother board with my old 6800 system. All went well and I became to busy to continue. Now some nine months (give or take a few more) later I m trying to bring up the rest of the system, still using the boards from Data Systems 68. I kept the 4 address I/O option and put my terminal at E004. So far I have not even been able to bring up the S-bug prompt with just the CPU an memory installed. Yes I did change the I/O address switch and I have memory continuous from 0000to DFFF.

Certainly many of your readers must have tried this system with varying degrees of success. Have I forgotten some simple little thing? Your help will be gratefully received. Thanks.

Respectfully

Bill

W. B. Jarzembski

P.S. I would have put this on disk but my system is down.

FLEX Equates

The listing of FLEX equates contains most of the storage locations, DOS user callable subroutines, and various dummy data structures and equates needed for proper 6809 assembly language programming. All equated values were taken from the TSC FLEX Programmer's Manual for the 6809 version of FLEX.

Any of you out there who have programmed in IBM 360/370 assembler know the use of a DSECT (Dummy Section). I have defined Dsects for an FCB (File Control Block) and a SIR (System Information Record). The format described by a Dsect may be associated with a particular area of storage. For example, to access the various fields within an FCB, an index register should contain the address of an FCB storage area. It is then just a matter of using the variables in the FCB Dsect, along with the index register, to access any field in the area.

```
Example:
          LDX   #SYSFCB   X-> FCB storage area
          LDD     #SIRTS    point to System Info
Record
          STD   FCBOP,X   set trk/sec in FCB
          LDA   #XRSS     get function code to read
          STA   FCBFC,X   save code in FCB
          JSR   FMSCAL    read the SIR from disk
          BCS   ERROR     branch if error
          LDY     #SIRFCB+FCBSB  point to SIR's
sector buffer
          LDD   SIRVOL,Y  get volume# of disk
            .
            .
            .
```

```
     *
     * FLEX Subroutine Linkages
     *
CD00 FLEX   EQU  $C000
CD00 COLDS  EQU  FLEX+$00 coldstart entry point
CD03 WARMS  EQU  FLEX+$03 warmstart entry point
CD06 RENTER EQU  FLEX+$06 DOS main loop re-entry point
CD09 INCH   EQU  FLEX+$09 input character
CD0C INCH2  EQU  FLEX+$0C input character
CD0F OUTCH  EQU  FLEX+$0F output character
CD12 OUTCH2 EQU  FLEX+$12 output character
CD15 GETCHR EQU  FLEX+$15 get character
CD18 PUTCHR EQU  FLEX+$18 put character
CD1B INBUF  EQU  FLEX+$1B input into line buffer
CD1E PSTRNG EQU  FLEX+$1E print string with crlf
CD21 CLASS  EQU  FLEX+$21 classify character
CD24 PCRLF  EQU  FLEX+$24 print CR and LF
CD27 NXTCH  EQU  FLEX+$27 get next buffer character
CD2A RSTRIO EQU  FLEX+$2A restore I/O vectors
CD2D GETFIL EQU  FLEX+$2D get file specification
CD30 LOAD   EQU  FLEX+$30 file loader
CD33 SETEXT EQU  FLEX+$33 set extension
```

```
0000 BIN    EQU  0         **
0001 TXT    EQU  1         **
0002 CMD    EQU  2         **
0003 BAS    EQU  3         **
0004 SYS    EQU  4         **
0005 BAK    EQU  5         **
0006 SCR    EQU  6         **
0007 DAT    EQU  7         **
0008 BAC    EQU  8         **
0009 DIR    EQU  9         **
000A PRT    EQU  10        **
000B OUT    EQU  11        **
CD36 ADDBX  EQU  FLEX+$36  add B-register to X-register
CD39 OUTDEC EQU  FLEX+$39  output decimal number
CD3C OUTHEX EQU  FLEX+$3C  output hexadecimal number
CD3F RPTERR EQU  FLEX+$3F  report error
CD42 GETHEX EQU  FLEX+$42  get hexadecimal number
CD45 OUTADR EQU  FLEX+$45  output hexadecimal address
CD48 INDEC  EQU  FLEX+$48  output decimal number
CD4B DOCMND EQU  FLEX+$4B  call DOS as a subroutine
CD4E STAT   EQU  FLEX+$4E  check terminal input status
     *
     * File Management System Entry Points
     *
C840 SYSFCB EQU  $C840     System FCB
D400 FMS    EQU  $D400     File Management System entry
D400 FMSINT EQU  FMS+$00   FMS Initialization
D403 FMSCLS EQU  FMS+$03   FMS close
D406 FMSCAL EQU  FMS+$06   FMS call


     *
     * Global Variables
     *
D409 FCBASE EQU  FMS+$09   FCB base pointer
D40B FCBCUR EQU  FMS+$0B   current FCB address
D435 FCBVER EQU  FMS+$35   verify flag
     *
     * DOS memory map
     *
C080 LNEBUF EQU  $C080     to $C0FF (128 byte line buf)
CC00 MAP    EQU  $CC00     start of map
CC00 BS     EQU  MAP+$00   TTYSET backspace char
CC01 DEL    EQU  MAP+$01   TTYSET delete character
CC02 EOL    EQU  MAP+$02   TTYSET end of line character
CC03 DEPTH  EQU  MAP+$03   TTYSET depth count
CC04 WIDTH  EQU  MAP+$04   TTYSET width count
CC05 NULL   EQU  MAP+$05   TTYSET null count
CC06 TAB    EQU  MAP+$06   TTYSET tab character
CC07 BSE    EQU  MAP+$07   TTYSET backspace echo character
CC08 EJECT  EQU  MAP+$08   TTYSET eject count
CC09 PAU    EQU  MAP+$09   TTYSET pause control
CC0A ESC    EQU  MAP+$0A   TTYSET escape character
CC0B SYDRV  EQU  MAP+$0B   system drive number
CC0C WKDRV  EQU  MAP+$0C   work drive number
CC0D SYSCR1 EQU  MAP+$0D   system scratch
CC0E SYDR   EQU  MAP+$0E   system date registers
CC11 LSTRM  EQU  MAP+$11   last terminator
CC12 UCTA   EQU  MAP+$12   user command table address
CC14 BUFPNT EQU  MAP+$14   line buffer pointer
CC16 ESCRR  EQU  MAP+$16   escape return register
CC18 CURC   EQU  MAP+$18   current character
CC19 PREVC  EQU  MAP+$19   previous character
CC1A CLN    EQU  MAP+$1A   current line number
CC1B LAO    EQU  MAP+$1B   loader address offset
CC1D TRFLG  EQU  MAP+$1D   transfer flag
CC1E TRADDR EQU  MAP+$1E   transfer address
CC20 FMSERR EQU  MAP+$20   error type
CC21 IOFLG  EQU  MAP+$21   special I/O flag
CC22 OSWTCH EQU  MAP+$22   output switch
CC23 ISWTCH EQU  MAP+$23   input switch
CC24 FOA    EQU  MAP+$24   file output address
CC26 FIA    EQU  MAP+$26   file input address
CC28 CMDFLG EQU  MAP+$28   command flag
CC29 COC    EQU  MAP+$29   current output column
CC2A SYSCR2 EQU  MAP+$2A   system scratch
CC2B MEMEND EQU  MAP+$2B   memory end
CC2D ENV    EQU  MAP+$2D   error name vector
```

```
CC2F FIEF   EQU  MAP+$2F   file input echo flag
CC30 SYSCR3 EQU  MAP+$30   system scratch
CC4E SYSCON EQU  MAP+$4E   system constants
CCC0 PRINIT EQU  MAP+$C0   printer initialize
CCD8 PRCHK  EQU  MAP+$D8   printer ready check
CCE4 POUT   EQU  MAP+$E4   printer output
CCF8 SYSCR4 EQU  MAP+$F8   system scratch

               *
               * Dsect for an FCB
               *
0000                     ORG  $0000
0000           FCBFC     RMB  1          function code
0001           FCBESB    RMB  1          error status byte
0002           FCBAS     RMB  1          activity status
          0001 ASREAD    EQU  1          **open for read
          0002 ASWRIT    EQU  2          **open for write
0003           FCBDN     RMB  1          drive number
0004           FCBNAM    RMB  8          file name
000C                     RMB  3          extension
000F           FCBFA     RMB  1          file attributes
          0080 FAWP      EQU  %10000000  **write protect
          0040 FADP      EQU  %01000000  **delete protect
          0020 FARP      EQU  %00100000  **read protect
          0010 FACP      EQU  %00010000  ** catalog protect
0010           FCBRS1    RMB  1          reserved for future use
0011           FCBSDA    RMB  2          starting disk addr of file
0013           FCBEDA    RMB  2          ending disk addr of file
0015           FCBFS     RMB  2          file size
0017           FCBFSM    RMB  1          file sector map indicator
          0000 FSMSEQ    EQU  0          **sequential file
          0002 FSMRAN    EQU  2          **random file
0018           FCBRS2    RMB  1          reserved for future use
          0019 FCBFCD    EQU  *          file creation date
0019           FCDMTH    RMB  1          **month
001A           FCDDAY    RMB  1          **day
001B           FCDYR     RMB  1          **year
001C           FCBLP     RMB  2          FCB list pointer
001E           FCBCP     RMB  2          trk/sec currently in sec buff
0020           FCBCRN    RMB  2          current record number
0022           FCBDI     RMB  1          data index
0023           FCBRI     RMB  1          random index
0024           FCBNWB    RMB  11         name work buffer
002F           FCBCDA    RMB  3          current directory address
0032           FCBFDD    RMB  3          first deleted dir ptr
0035           FCBSCR    RMB  11         scratch bytes
003B                     ORG  FCBSCR+6
003B           FCBSCF    RMB  1          space compression flag
          0000 SCFSC     EQU  $00        **perform space compr.
          00FF SCFNSC    EQU  $FF        **perform no space compr.
0040                     ORG  FCBSCR+11
          0040 FCBSB     EQU  *          sector buffer
0040           SBLINK    RMB  2          next trk/sector in chain
0042           SBRS1     RMB  2          reserved for future use
0044           SBDATA    RMB  252        data storage
          0140 FCBLEN    EQU  *          length of FCB

     *
     * Function Codes
     *
0000 XRWNB  EQU  0         read/write next byte/char
0001 XOREAD EQU  1         open for read
0002 XOWRIT EQU  2         open for write
0003 XOUPDT EQU  3         open for update
0004 XCLOSE EQU  4         close file
0005 XREWND EQU  5         rewind file
0006 XODIR  EQU  6         open directory
0007 XGIR   EQU  7         get information record
0008 XPIR   EQU  8         put information record
0009 XRSS   EQU  9         read single sector
000A XWSS   EQU  10        write single sector
000B XRES1  EQU  11        reserved for future use
000C XDELET EQU  12        delete file
000D XRENAM EQU  13        rename file
000E XRES2  EQU  14        reserved for future use
000F XNSS   EQU  15        next sequential sector
0010 XOSIR  EQU  16        open system info rec
```

```
                0011  XGRB    EQU    17         get random byte from sector
                0012  XPRB    EQU    18         put random byte in sector
                0013  XRES3   EQU    19         reserved for future use
                0014  XFND    EQU    20         find next drive
                0015  XPOSN   EQU    21         position to record n
                0016  XBOR    EQU    22         backup one record
                      *
                      * Dsect for a SIR
                      *
0000                          ORG    $0000
0000                          RMB    16         16 byte header
0010                  SIRNAM  RMB    8          volume name
0018                          RMB    3          extension
001B                  SIRVOL  RMB    2          volume number
001D                  SIRFSB  RMB    2          beginning of free chain
001F                  SIRFSE  RMB    2          end of free chain
0021                  SIRFSS  RMB    2          # sectors in free chain
                0023  SIRCRE  EQU    *          creation date of disk
0023                  SIRMTH  RMB    1          **month
0024                  SIRDAY  RMB    1          **day
0025                  SIRYR   RMB    1          **year
0026                  SIRMTS  RMB    2          maximum trk/sec available
                0028  SIRLEN  EQU    *          SIR length

      *
      * Miscellanious equates
      *
0003  SIRTS    EQU    $0003     trk/sec of SIR
0005  DIRTS    EQU    $0005     trk/sec of 1st node in dir
      *
0004  EOT      EQU    4         end of text delimiter
000A  CRLF     EQU    $000A     carriage return, line feed
000D  CR       EQU    $0D       carriage return
000A  LF       EQU    $0A       line feed
0007  BELL     EQU    $07       bell
0020  SP       EQU    $20       space
      *
0000  URAM     EQU    $0000     to $BFFF (User RAM area)
C000  STKA     EQU    $C000     to $C07F (SP inited to C07F)
C100  UCA      EQU    $C100     to $C6FF (Util Cmd Area)
C700  SPS      EQU    $C700     to $C83F (Schdlr & Spooler)
C980  SFA      EQU    $C980     to CBFF (System Files Area)
CC00  DOS      EQU    $CC00     to D3FF (DOS)
F700  CLOCK    EQU    $F700     real time clock (DOS)
DE00  DDRV     EQU    $DE00     to DFFF (Disk Drivers)
               OPT    LIS


SYMBOL TABLE:

ADDBX  CD36   ASREAD 0001   ASWRIT 0002   BAC    0008   BAK    0005
BAS    0003   BELL   0007   BIN    0000   BS     CC00   BSE    CC07
BUFPNT CC14   CLASS  CD21   CLN    CC1A   CLOCK  F700   CMD    0002
CMDFLG CC28   COC    CC29   COLDS  CD00   CR     000D   CRLF   000A
CURC   CC18   DAT    0007   DDRV   DE00   DEL    CC01   DEPTH  CC03
DIR    0009   DIRTS  0005   DOCMND CD4B   DOS    CC00   EJECT  CC08
ENV    CC20   EOL    CC02   EOT    0004   ESC    CC0A   ESCRR  CC16
FACP   0010   FADP   0040   FARP   0020   FAWP   0080   FCBAS  0002
FCBASE D409   FCBCDA 002F   FCBCP  001E   FCBCRN 0020   FCBCUR D40B
FCBDI  0022   FCBDN  0003   FCBEDA 0013   FCBESB 0001   FCBFA  000F
FCBFC  0000   FCBFMO 0019   FCBFDO 0032   FCBFS  0015   FCBFSM 0017
FCBLEN 0140   FCBLP  001C   FCBNAM 0004   FCBMAP 0024   FCBRI  0023
FCBRS1 0010   FCBRS2 0018   FCBSB  0040   FCBSCF 003B   FCBSCR 0035
FCBSDA 0011   FCBVER D435   FCDDAY 001A   FCDMTH 0019   FCDYR  001B
FIA    CC26   FIEF   CC2F   FLEX   CD00   FMS    D400   FMSCAL D406
FMSCLS D403   FMSERR CC20   FMSINT D400   FOA    CC24   FSMRAN 0002
FSMSEQ 0000   GETCHR CD15   GETFIL CD2D   GETHEX CD42   INBUF  CD1B
INCH   CD09   INCH2  CD0C   INDEC  CD48   IOFLG  CC21   ISWTCH CC23
LAO    CC1B   LF     000A   LNEBUF C080   LOAD   CD30   LSTRM  CC11
MAP    CC00   MEMEND CC2B   NULL   CC05   NXTCH  CD27   OSWTCH CC22
OUT    0008   OUTADR CD45   OUTCH  CD0F   OUTCH2 CD12   OUTDEC CD39
OUTHEX CD3C   PAU    CC09   PCRLF  CD24   POUT   CCE4   PRCHK  CCD8
PREVC  CC19   PRINIT CCC0   PRT    000A   PSTRNG CD1E   PUTCHR CD18
RENTER CD06   RPTERR CD3F   RSTRIO CD2A   SBDATA 0044   SBLINK 0040
SBRS1  0042   SCFWSC 00FF   SCFSC  0000   SCR    0006   SETEXT CD33
SFA    C980   SIRCRE 0023   SIRDAY 0024   SIRFSB 001D   SIRFSE 001F
SIRFSS 0021   SIRLEN 0028   SIRMTH 0023   SIRMTS 0026   SIRNAM 0010
SIRTS  0003   SIRVOL 001B   SIRYR  0025   SP     0020   SPS    C700
STAT   CD4E   STKA   C000   SYDR   CC0E   SYDRV  CC08   SYS    0004
SYSCON CC4E   SYSCR1 CC0D   SYSCR2 CC2A   SYSCR3 CC30   SYSCR4 CCF8
SYSFCB C840   TAB    CC06   TRADDR CC1E   TRFLG  CC1D   TXT    0001
UCA    C100   UCTA   CC12   URAM   0000   WARMS  CD03   WIDTH  CC04
WKDRV  CC0C   XBOR   0016   XCLOSE 0004   XDELET 000C   XFND   0014
XGIR   0007   XGRB   0011   XNSS   000F   XODIR  0006   XOREAD 0001
XOSIR  0010   XOUPDT 0003   XOWRIT 0002   XPIR   0008   XPOSN  0015
XPRB   0012   XRENAM 0000   XRES1  000B   XRES2  000E   XRES3  0013
XREWND 0005   XRSS   0009   XRWNB  0000   XWSS   000A
```

DAVID V. OADBY
2, LUPIN CLOSE
HINCKLEY
LEICESTERSHIRE LE10 2UJ
ENGLAND

JUST A COUPLE OF PROGRAMS THAT MAY BE USEFUL TO FELLOW 68XX USERS. I'M SORRY THERE IS NO LOWER CASE BUT MY USED ADM3 HASNT GOT A LOWER CASE GENERATOR FITTED YET.. [ANYONE GOT A MANUAL ?].

THE FIRST SET OF PROGRAMS IS A PASCAL PROGRAM [LUCIDATA] WHICH DOES DIRECT INPUT-OUTPUT WITH THE TERMINAL USING AN EXTERNAL PROCEDURE. IF NOTHING ELSE IT SHOWS THAT IT WORKS. ACTUALLY IT IS PART OF AN ON-SCREEN WORD PROCESSING SYSTEM THAT I AM DEVELOPING..

IF YOU ARE A FLEX09 USER AND YOU HAVE ALWAYS WANTED A FUNCTION KEY FACILITY THEN READ ON. THERE ARE TWO PROGRAMS AND THEY WORK TOGETHER. THE FKEY CODE OVERLAYS THE FLEX INPUT VECTORS AND INTERCEPTS ALL KEY INPUTS IN ORDER TO TRAP THE FUNCTION KEY REQUEST (CURRENTLY THE TAB KEY).

THE FUNLOAD PROGRAM ALLOWS YOU TO LOAD PRESET FUNCTION KEY VALUES FROM A TEXT FILE. THE COMMENTEO CODE SHOULD PROVIDE ALL THE OTHER DETAILS..

FINALLY I HAVE JUST JOINED A COMPANY WHICH IS SELLING A 68000 BASED MACHINE SO ITS MOTOROLA ALL THE WAY.....

```
   0 PROGRAM KEYIO;
   0  (*
   0     AUTHOR   : D.V.GOADBY
   0     CREATE   : 28/ /82
   0     EDIT     : 25/6/82
   0     FILENAME: LINEDIA
   0     VERSION : 1.2 - [ LUCIDATA PASCAL V 3.9 ]
   0
   0     THIS PROGRAM ALLOWS DIRECT KEYBOARD INPUT/OUTPUT
   0     USING AN EXTERNAL PROCEDURE KEYIO. ALL KEYBOARD
   0     CODES ARE RETURNED TO THE PROGRAM AND THIS ALLOWS
   0     THE PROGRAMMER TO USE CONTROL CODES WHICH ARE NOT
   0     NORMALLY PASSED BACK TO THE PROGRAM.
   0     KEYIO IS CALLED WITH A FUNCTION CODE WHICH DETERMINES THE
   0     ACTION OF THE EXTERNAL PROCEDURE. ALTHOUGH ONLY READ,WRITE
   0     AND ECHO ARE IMPLEMENTED MANY OTHER POSSIBILITIES EXIST.
   0  *)
   0
   0 CONST
   0     EXTKEY =$F200; (* ADDRESS OF EXTERNAL PROCEDURE *)
   0     FREAD  =$01; (* READ KEYBOARD *)
   0     FWRITE =$02; (* WRITE TO SCREEN *)
   0     ECHOFF =$03; (* TURN ECHO OFF *)
   0     ECHOON =$04; (* TURN ECHO ON *)
   0     FINISH =CHR($18); (* TERMINATE CHARACTER *)
   0
   0 VAR
   0     CHR1:CHAR;
   0
   0 PROCEDURE KEYIO(VAR CHR:CHAR;FUNCODE:BYTE);
   4 EXTERNAL EXTKEY;
   4
   4 (* MAIN PROGRAM - TO SEE HOW IT WORKS *)
   4 BEGIN
   8     WRITELN(' KEYIO TEST PROGRAM TYPE CTRL X TO TERMINATE.');
  64     KEYIO(CHR1,ECHOFF); (* TURN OFF ECHO *)
  84     REPEAT
  84        WRITELN;
  88        WRITE(' INPUT A CHARACTER :');
 116        KEYIO(CHR1,FREAD);
 136        WRITE(' CHARACTER IS:');
 156        KEYIO(CHR1,FWRITE);
 176        WRITELN (' CHAR VAL IS ',ORD(CHR1));
 212     UNTIL CHR1=FINISH;
 224     WRITE('NOW ENTER A CHARACTER :');
 256     KEYIO(CHR1,ECHOON); (* RESTORE ECHO*)
 276     KEYIO(CHR1,FREAD);
 296     WRITELN('<-- IF ECHO WAS RESTORED THEN LAST INPUT SHOULD BE SEEN');
 364 END.
 364 BYTES
END OF PASS 1
END OF PASS 2
OK  TO RUN
```

```
                        NAM     KEYIO

                * AUTHOR   : D.V.COADBY
                * CREATE   : 24/6/82
                * EDIT     : 25/6/82
                * FILENAME : INOUT1.ASM
                * VERSION  : 1.1 - 1 FOR USE WITH LUCIDATA PASCAL V 2.9 & P8

                * THIS CODE IS THE EXTERNAL PROCEDURE KEYIO
                * CALLED EXTERNALLY FROM THE PASCAL PROGRAM.
                * THE FORMAT IS EXPLAINED IN THE PASCAL LISTING.

                * LUCIDATA PARAMETERS ARE PLACED ON A STACK POINTED TO
                * BY A TERM CALLED MARKUS. THE FIRST 6 BYTES ARE THE
                * ACTIVATION RECORD AND ARE NOT TO BE TOUCHED.
                * MARKUS+6 CONTAINS THE POINTER TO THE CHARACTER VARIABLE.
                * MARKUS+8 CONTAINS THE ACTUAL FUNCTION CODE .

                * EQUATES

          F200  PRGLOC  EQU     $F200       LOCATION OF THIS PROGRAM
          0144  MARKUS  EQU     $144        BASE OF STACK FOR USER CALL
          F06C  ECHO    EQU     $F06C       CONTAINS ADDRESS OF ECHO ROUTINE 1
          F05E  ECHOAD  EQU     $F05E       ACTUAL ADDRESS OF ECHO ROUTINE
          FD44  PSYINP  EQU     $FD44       PSYMON INPUT ROUTINE
          FD58  PSYOUT  EQU     $FD58       PSYMON OUTPUT ROUTINE

                * FUNCTION CODE EQUATES

          0001  READ    EQU     1           READ KEYBOARD
          0002  WRITE   EQU     2           WRITE TO SCREEN
          0003  ECHOFF  EQU     3           TURN OFF ECHO
          0004  ECHON   EQU     4           RESTORE ECHO

F200                    ORG     PRGLOC

          F200  ENTRY   EQU     *
F200 BE    0144         LDX     MARKUS      GET STACK POINTER
F203 A6    08           LDA     8,X         GET FUNCTION CODE
F205 81    01           CMPA    £READ
F207 27    1A           BEQ     READ1
F209 81    02           CMPA    £WRITE
F20B 27    1D           BEQ     WRITE1
F20D 81    03           CMPA    £ECHOFF
F20F 26    09           BNE     ECHOON
F211 108E  0000         LDY     £0
F215 10BF  F06C         STY     ECHO        CLEAR ECHO ADDRESS
F219 39                 RTS
F21A 108E  F05E ECHOON  LDY     £ECHOAD
F21E 10BF  F06C         STY     ECHO        PUT BACK ECHO ADDRESS
F222 39                 RTS                 IGNORE OTHER CODES

                * READ ROUTINE

F223 BD    FD44 READ1   JSR     PSYINP      GET KEY VALUE
F226 A7    98 06        STA     [6,X]       PUT RESULT ON STACK
F229 39                 RTS

                * WRITE ROUTINE

F22A A6    98 06 WRITE1 LDA     [6,X]       GET DATA
F22D BD    FD58         JSR     PSYOUT
F230 39                 RTS


                        END

0 ERROR(S) DETECTED

SYMBOL TABLE:

ECHO    F06C   ECHOAD F05E   ECHOFF 0003   ECHON  0004   ECHOON F21A
ENTRY   F200   MARKUS 0144   PRGLOC F200   PSYINP FD44   PSYOUT FD58
READ    0001   READ1  F223   WRITE  0002   WRITE1 F22A
```

```
 *********************************************************
 * PROGRAM  : FUNCTION KEYS                              *
 * AUTHOR   : DAVID V. COADBY                            *
 * CREATE   : 9/9/80                                     *
 * EDIT     : 19/3/81                                    *
 * FILENAME : 14.FKEY09.TXT                              *
 * VERSION  : 1.3 FLEX09/PSYMON                          *
 *********************************************************

 * THIS PROGRAM ALLOWS THE USER TO DEFINE 9 FUNCTION
 * KEY STRINGS.
 * THE PROGRAM INTERCEPTS THE NORMAL FLEX INPUT
 * FROM THE KEYBOARD AND CHECKS FOR THE TAB KEY.
 * THE TAB KEY WAS MY CHOICE FOR THE TRIGGER BUT YOU
 * CAN CHANGE IT TO ANYTHING YOU LIKE.
 * IF THE TAB CHARACTER IS DETECTED THEN THE NEXT
 * CHARACTER IS CHECKED FOR A VALUE OF 1-9 SIGNIFYING
 * A FUNCTION KEY REQUEST. IF THE VALUE IS 0 THEN THIS
 * INDICATES AN EDIT REQUEST. THE EDIT FUNCTION ALLOWS
 * THE USER TO DISPLAY AND ALTER ANY OF THE FUNCTION
 * STRINGS CURRENTLY HELD. THE SYSTEM HAS A STANDARD
 * SET OF COMMANDS INITIALLY.
 * THERE ARE 9 FUNCTION KEYS AND EACH KEY CAN HAVE A
 * 24 CHARACTER STRING ASSIGNED TO IT. THE STRING LENGTH CAN
 * BE ALTERED BY CHANGING THE CONSTANT LINEL.
 * IF THE STRING IS TERMINATED WITH A \ THEN FURTHER USER
 * INPUT IS WAITED FOR. OTHERWISE A COMPLETE STRING PLUS
 * A CARRIAGE RETURN IS SENT TO FLEX.
 * THE ACTUAL FUNCTION KEY CONSISTS OF A TWO KEY
 * SEQUENCE TAB-NUM .THE TAB KEY (09) TRIGGERS
 * A TRAP WHICH CHECKS THE FOLLOWING KEY VALUE.
 * IF THE KEY IS 1 TO 9 THEN THE PARTICULAR FUNCTION
 * DESCRIBED BY THAT KEY IS SENT TO FLEX.
 * A KEY VALUE OF 0 INDICATES THAT THE
 * USER WISHES TO EDIT ONE OF THE PREDESCRIBED
 * FUNCTIONS. AFTER A 0 THE NEXT CHARACTER SHOULD
 * BE 1-9 AND THE FUNCTION VALUE WILL BE DISPLAYED.
 * THE USER THEN ENTERS N FOR NO ACTION OR R FOR
 * REPLACE FOLLOWED BY THE NEW STRING.
 * A COMPANION PROGRAM CALLED FUNLOAD CAN BE USED
 * TO PRELOAD THE TABLE WITH OTHER STRING SETS.
 * THE USER CAN HAVE A DIFFERENT SET OF FUNCTION KEYS
 * FOR DIFFERENT APPLICATIONS EG BASIC,ASSEMBLER ETC.

 * MY SYSTEM HAS A FULL COMPLEMENT OF MEMORY SO
 * I LOCATED THE CODE BELOW $C000. A CLOSE
 * LOOK AT THE ORG STATEMENTS SHOULD REVEAL HOW
 * I DID IT. (HINT: I ASSEMBLED IT TWICE).


        * SPECIFIC EQUATES

0018  LINEL   EQU     24          LINE LENGTH ( INCL CR)
0009  TRIGGR  EQU     $09         TAB CHARACTER FOR TRIGGER

        * FLEX EQUATES
```

```
             CC00  BACK    EQU     $CC00      BACKSPACE ON TERMINAL
             CD03  WARMS   EQU     $CD03      WARM START
             CD09  FINP1   EQU     $CD09      FLEX CALLS EIAC HERE.
             CD0C  FINP2   EQU     $CD0C      AND HERE.....
             FD58  OUTPUT  EQU     $FD58      PSYMON OUTPUT ROUTINE
             FD44  INPUT   EQU     $FD44      PSYMON INPUT ROUTINE

             FDA2  CRLF    EQU     $FDA2      PSYMON CRLF

                   * FLEX OVERLAYS

CD0A                       ORG     FINP1+1    FLEX INPUT VECTOR
CD0A BE56                  FDB     START
CD0D                       ORG     FINP2+1
CD0D BE56                  FDB     START

                   * ORIGIN AS HIGH AS POSSIBLE

             01AA  PROGL1  EQU     PROGEND-START LENGTH OF CODE
             01AA  PROGL   EQU     $01AA      FROM ABOVE
BE56                       ORG     $C000-PROGL PUT RIGHT OUT OF THE WAY

                   * MAIN PROGRAM

BE56 34    04      START   PSHB
BE58 BF    BE70            STX     XSAVE
BE5B 7D    BE75            TST     FLAG       IS DIVERT STILL ON
BE5E 26    27              BNE     FINPT      GET BUFFER INPUT
BE60 BD    FD44    START1  JSR     INPUT      GET CHARACTER AS NORMAL
BE63 81    09              CMPA    £$09       TAB ? ** CHANGE T IS IF YOU LIKE
BE65 27    0F              BEQ     TRAP1
BE67 BF    BE72    RETURN  STX     XSAVE1     SAVE BUFFER POINTER
BE6A 35    04              PULB
BE6C BE    BE70            LDX     XSAVE
BE6F 39                    RTS
BE70               XSAVE   RMB     2
BE72               XSAVE1  RMB     2
BE74               FNUM    RMB     1          FUNCTION KEY NUMBER
BE75 00            FLAG    FCB     0          DIVERT FLAG

             BE76  TRAP1   EQU     *
                   * JSR BACKSPACE  NEEDED FOR SOME TERMINALS
BE76 BD    FD44            JSR     INPUT      GET NEXT CHARACTER
BE79 BD    BF1C            JSR     BSPACE
BE7C 81    30      EDITCK  CMPA    £'0        EDIT REQUIRED ?
BE7E 27    25              BEQ     EDIT
BE80 8D    7C              BSR     VALID      CHECK 1-9 AND GET ADDRESS OF CMD.
BE82 25    DC              BCS     START1     INVALID
BE84 73    BE75            COM     FLAG       SET DIVERSION
BE87 BE    BE72    FINPT   LDX     XSAVE1
BE8A A6    84              LDAA    0,X        GET NEXT CHARACTER
BE8C 81    0D              CMPA    £$0D       END OF STRING ?
BE8E 26    05              BNE     CHKNIT
BE90 7F    BE75    FUNOFF  CLR     FLAG       SET TO NORMAL
BE93 20    D2              BRA     RETURN
BE95 81    5C      CHKNIT  CMPA    £'\        USER INPUT REQUIRED ?
BE97 26    05              BNE     DISPL1
BE99 7F    BE75            CLR     FLAG       SET TO NORMAL
BE9C 20    C2              BRA     START1     GET USER INPUT
BE9E BD    FD58    DISPL1  JSR     OUTPUT     SEND CHARACTER TO TERMINAL
BEA1 30    01              INX                POINT TO NEXT ENTRY
BEA3 20    C2              BRA     RETURN

BEA5 BD    FD44    EDIT    JSR     INPUT      GET FUNCTION NO.
BEA8 8D    72              BSR     BSPACE
BEAA 8D    52              BSR     VALID      SEE IF 1-9 AND GET ADDRESS
BEAC 25    B2              BCS     START1     ERROR
BEAE BF    BE72            STX     XSAVE1     SAVE ENTRY VALUE

                   * DISPLAY FUNCTION KEY CONTENTS

BEB1 86    46      DISPLAY LDAA    £'F

BEB3 BD    FD58            JSR     OUTPUT
BEB6 B6    BE74            LDAA    FNUM       GET NUMBER
BEB9 BD    FD58            JSR     OUTPUT
BEBC 86    3D              LDAA    £'=
BEBE BD    FD58            JSR     OUTPUT
BEC1 BE    BE72            LDX     XSAVE1
BEC4 A6    80      PLOOP   LDAA    0,X+       GET CHARACTER
BEC6 81    0D              CMPA    £$0D       EOL ?
BEC8 27    09              BEQ     CONEIT
BECA BD    FD58            JSR     OUTPUT
BECD 20    F5              BRA     PLOOP      PRINT THE REST
BECF 86    0D      START2  LDAA    £$0D       SEND CR TO FLEX
BED1 20    94              BRA     RETURN
                   * NOW CHECK FOR R
BED3 86    3A      CONEIT  LDAA    £':        SEPERATOR
BED5 BD    FD58            JSR     OUTPUT
BED8 BD    FD44            JSR     INPUT      GET OPTION
BEDB 81    52              CMPA    £'R        REPLACE ?
BEDD 26    F0              BNE     START2
                   * ENTER NEW STRING
BEDF 86    3D              LDAA    £'=        SIGNIFY START OF STRING
BEE1 BD    FD58            JSR     OUTPUT
BEE4 BE    BE72            LDX     XSAVE1     GET TABLE ENTRY
BEE7 C6    17              LDAB    £LINEL-1   LENGTH OF BUFFER-1
BEE9 BD    FD44    EDLOOP  JSR     INPUT      GET NEXT CHARACTER
BEEC A7    80              STAA    0,X+       SAVE ASSIGNED STRING IN TABLE
BEEE 81    0D              CMPA    £$0D       FINISHED ?
BEF0 27    07              BEQ     DONE
BEF2 5A                    DECB
BEF3 26    F4              BNE     EDLOOP     GET SOME MORE
BEF5 86    0D              LDAA    £$0D       FORCE CR
BEF7 A7    80              STAA    0,X+
BEF9 BD    FDA2    DONE    JSR     CRLF
BEFC 20    B3              BRA     DISPLAY    SHOW NEW ENTRY AND WAIT FOR R OR N

                   * THIS ROUTINE VALIDATES THE FKEY NUMBER 1-9 AND
                   * RETURNS THE ACTUAL ADDRESS OF THE STRING IN X
                   * IF VALIDATION FAILS THEN THE CARRY BIT IS SET.

BEFE B7    BE74    VALID   STAA    FNUM       SAVE FUNCTION NUMBER
BF01 81    30              CMPA    £'0
BF03 23    14              BLS     ERROR
BF05 81    39              CMPA    £'9
BF07 22    10              BHI     ERROR
BF09 80    31              SUBA    £'1        SET TO BINARY-1
BF0B 8E    BF27            LDX     £TABLE
                   * MULTIPLY A BY LINEL TO INDEX INTO TABLE
                   * THANKS FOR 6809
                   * THEN ADD TO X TO POINT TO ENTRY.
BF0E C6    18              LDB     £LINEL     GET LINE LENGTH
BF10 3D                    MUL                A*B->D
BF11 30    8B              LEAX    D,X        X=X+D
BF13 BF    BE72            STX     XSAVE1
BF16 1C    FE              CLC                OK
BF18 39                    RTS
BF19 1A    01      ERROR   SEC                NOT OK
BF1B 39                    RTS
BF1C 34    02      BSPACE  PSHA
```

```
BF1E 86    CC00            LDAA   BACK       GET BACKSPACE CHAR
BF21 BD    FD58            JSR    OUTPUT
BF24 35    02              PULA
BF26 39                    RTS

                  * TABLE OF PRESET COMMANDS

           BF27   TABL     EQU    *
BF27 44 49 52              FCC    /DIR/
BF2A 0D                    FCB    $D
BF3F                       ORG    1*LINEL+TABLE
BF3F 4C 49 53 54           FCC    /LIST,\/
BF43 2C 5C
BF45 0D                    FCB    $D
BF57                       ORG    2*LINEL+TABLE
BF57 50 44 45 4C           FCC    /PDEL,\/
BF5B 2C 5C
BF5D 0D                    FCB    $D
BF6F                       ORG    3*LINEL+TABLE
BF6F 45 44 49 54           FCC    /EDIT,\/
BF73 2C 5C
BF75 0D                    FCB    $D
BF87                       ORG    4*LINEL+TABLE
BF87 41 53 4D 42           FCC    /ASMB,\/
BF8B 2C 5C
BF8D 0D                    FCB    $D
BF9F                       ORG    5*LINEL+TABLE
BF9F 43 4F 50 59           FCC    /COPY,0,1,\/
BFA3 2C 30 2C 31
BFA7 2C 5C
BFA9 0D                    FCB    $D
BFB7                       ORG    6*LINEL+TABLE

BFB7 44 49 52 2C           FCC    /DIR,0,.CMD/
BFBB 30 2C 2E 43
BFBF 4D 44
BFC1 0D                    FCB    $D
BFCF                       ORG    7*LINEL+TABLE
BFCF 4D 41 50 2C           FCC    /MAP,\/
BFD3 5C
BFD4 0D                    FCB    $D
BFE7                       ORG    8*LINEL+TABLE
BFE7 42 41 53 49           FCC    /BASIC/
BFEB 43
BFEC 0D                    FCB    $D
BFFF                       ORG    9*LINEL+TABLE
BFFF 00                    FCB    $0         DUMMY END
           C000   PROGEND  EQU    *          SHOULD BE $FFF HEX
                  * NOW TAP MEMEND IN FLEX TO ALLOW FOR FKEY
CC2B                       ORG    $CC2B      MEMEND
CC2B BE55                  FDB    START-1
                           END    FWARMS

0 ERROR(S) DETECTED

SYMBOL TABLE:

BACK   CC00   BSPACE BF1C   CHENIT BE95   CRLF   FDA2   DISPL1 BE9E
DISPLA BEB1   DONE   BEF9   EDIT   BEA5   EDITCK BE7C   COLOOP BEE9
EDNEXT BED3   ERROR  BF19   FINP1  CD09   FINP2  CD0C   FINPT  BEB7
FLAG   BE75   FNUM   BE74   FUNCRT BEE0   FWARMS CD03   INPUT  FD44
LINEL  0018   OUTPUT FD58   PLOOP  BEC4   PROGEN C000   PROGL  01AA
PROGL1 01AA   RETURN BE67   START  BE56   START1 BE60   START2 BECF
TABLE  BF27   TRAP1  BE76   TRIBGR 0009   VALID  BEFE   XSAVE  BE70
XSAVE1 BE72
```

```
                           NAM    FUNLOAD FLEX09 VERSION

                           OPT    PAG

                  ****************************************************
                  * PROGRAM  : FUNCTION KEY  LOADER                  *
                  * AUTHOR   : DAVID V. GOADBY                       *
                  * CREATE   : 9/8/80                                *
                  * EDIT     : 12/12/80                              *
                  * FILENAME : 14.FUNLOAD9.TXT                       *
                  * VERSION  : 1.3 FLEX09/FKEY09 V 1.3               *
                  ****************************************************

                  * THIS PROGRAM RELOADS THE STRING TABLE IN THE
                  * FUNCTION KEY MODULE WITH A SET OF PREDEFINED
                  * STRINGS FROM A TEXT FILE.
                  *
                  * COMMAND FORMAT: FUNLOAD,TEXTFILENAME
                  *
                  * IF ANY LINE IS TOO LONG OR THERE ARE LESS THAN 9
                  * ENTRIES ON THE FILE THEN AN ERROR MESSAGE IS DISPLAYED.

                  * THE FILE CAN BE CREATED USING THE EDITOR AND
                  * CONSISTS OF ONE LINE PER KEY STARTING AT KEY 1

                  * SPECIAL EQUATES ( REFER TO FKEY LISTING )

           BF27   TABLE    EQU    $BF27      TABLE ADDRESS IN FUNCTION KEY MODU
           0018   LINEL    EQU    24         LINE LENGTH

                  * FLEX EQUATES

           C840   FCB      EQU    $C840      SYSTEM FCB
           D406   FMS      EQU    $D406      FLEX ENTRY
           CD03   WARMS    EQU    $CD03
           CD2D   GETFIL   EQU    $CD2D      FILE SPEC
           CD3F   RPTERR   EQU    $CD3F      PRINT ERROR
           D403   FMSCLS   EQU    $D403      CLOSE ALL FILES
           CD33   SETEXT   EQU    $CD33      SET EXTENSION
           CD1E   STRING   EQU    $CD1E      PRINT STRING

C100                       ORG    $C100      TRANSIENT AREA

C100 20    01     START    BRA    START1
C102 03           VN       FCB    3          VERSION
C103 8E    C840   START1   LDX    #FCB       USE SYSTEM FCB
C106 BD    CD2D            JSR    GETFIL     GET FILE NAME
C109 25    69              BCS    ERROR      DUFF NAME
C10B 86    01              LDA    #1         OPEN FOR READ
C10D A7    84              STA    ,X         PUT INTO FCB
C10F BD    CD33            JSR    SETEXT     SET TO .TXT
C112 BD    D406            JSR    FMS        OPEN FILE
C115 26    5D              BNE    ERROR
                  * SETUP TABLE PARAMETERS
C117 8E    BF27            LDX    #TABLE     GET TABLE LOCATION
C11A BF    C1B6            STX    CURPOS     1ST CHAR OF 1ST LINE
C11D 108E  00D8            LDY    #9*LINEL   LENGTH OF TABLE

C121 10BF  C1B4            STY    EOFTAB     END OF TABLE +1
C125 8E    BF27            LDX    #TABLE
C128 C6    18              LDB    #LINEL     LENGTH OF SINGLE ENTRY
C12A 3A                    ABX
C12B BF    C1B2            STX    EOFLIN     END OF FIRST LINE +1
C12E C6    09              LDB    #9         LINE COUNT
C130 F7    C1B8            STB    LINECT
                  * READ DATA FROM DISK INTO TABLE
C133 8E    C840   READLP   LDX    #FCB       POINT TO FCB
C136 BD    D406            JSR    FMS        GET CHAR
C139 26    42              BNE    ERROR1
C13B BE    C1B6   STORLP   LDX    CURPOS     NEXT POSITION
C13E BC    C1B2            CPX    EOFLIN     END OF LINE +1 ?
C141 27    0B              BEQ    LINEND
C143 A7    80              STA    ,X+        PUT IN TABLE
C145 BF    C1B6            STX    CURPOS     STORE NEXT POSITION
C148 81    0D              CMPA   #$D        CR ?
C14A 26    E7              BNE    READLP
C14C 20    ED              BRA    STORLP     FILL TO END OF LINE WITH CR
                  * END OF LINE PROCESSING
C14E A6    1F     LINEND   LDA    -1,X       GET LAST CHAR
C150 81    0D              CMPA   #$D        SHOULD BE CR..
C152 26    31              BNE    ERROR2
C154 BE    C1B2            LDX    EOFLIN     GET START OF NEXT LINE
C157 BF    C1B6            STX    CURPOS     SET READY FOR NEXT LINE
C15A C6    18              LDB    #LINEL
C15C 3A                    ABX
C15D BF    C1B2            STX    EOFLIN
C160 7A    C1B8            DEC    LINECT     BUMP COUNT
C163 26    CE              BNE    READLP     OK TO CONTINUE
C165 86    04     ENDIT    LDA    #4         CLOSE FILE
C167 8E    C840            LDX    #FCB
C16A A7    84              STA    ,X
C16C BD    D406            JSR    FMS
C16F 26    03              BNE    ERROR      COULD BE SERIOUS !
C171 7E    CD03            JMP    WARMS

                  * ERROR PROCESSING

C174 BD    CD3F   ERROR    JSR    RPTERR     REPORT ERROR
C177 BD    D403            JSR    FMSCLS     CLOSE ALL FILES
C17A 7E    CD03            JMP    WARMS      RETURN TO FLEX
                  * THE FOLLOWING ARE SOFT ERRORS
C17D 8E    C18A   ERROR1   LDX    #MSG1
C180 BD    CD1E   ERREXT   JSR    STRING
C183 20    E0              BRA    ENDIT
C185 8E    C19E   ERROR2   LDX    #MSG2
C188 20    F6              BRA    ERREXT

                  * ERROR MESSAGE STRING

C18A 4C 45 53 53  MSG1     FCC    /LESS THAN 9 ENTRIES/
C18E 20 54 48 41
C192 4E 20 39 20

C196 45 4E 54 52
C19A 49 45 53
C19D 04                    FCB    4
C19E 4C 49 4E 45  MSG2     FCC    /LINE >24 CHARACTERS/
C1A2 20 3E 32 34
C1A6 20 43 48 41
C1AA 52 41 43 54
C1AE 45 52 53
C1B1 04                    FCB    4

                  * DATA AREA

C1B2              EOFLIN   RMB    2          END OF CURRENT LINE
C1B4              EOFTAB   RMB    2          END OF TABLE
C1B6              CURPOS   RMB    2          NEXT TABLE ENTRY
C1B8              LINECT   RMB    1          LINES COUNT

                           END    START

0 ERROR(S) DETECTED

SYMBOL TABLE:

CURPOS C1B6   ENDIT  C165   EOFLIN C1B2   EOFTAB C1B4   ERREXT C180
ERROR  C174   ERROR1 C17D   ERROR2 C185   FCB    C840   FMS    D406
FMSCLS D403   GETFIL CD2D   LINECT C1B8   LINEL  0018   LINEND C14E
MSG1   C18A   MSG2   C19E   READLP C133   RPTERR CD3F   SETEXT CD33
START  C100   START1 C103   STORLP C13B   STRING CD1E   TABLE  BF27
VN     C102   WARMS  CD03
```

Mr. Don Williams
68 Micro Journal
5900 Cassandra Smith
P.O.Box 849
Hixson, Tn. 37343

3033 N.San Gabriel,#26
Rosemead, Ca. 91770
19 Oct. 1984
818-280-6377

Dear Mr. Williams,

I read with interest, and some amusement, your review of "Fire in the Valley". It is not too surprising that the book is so inaccurate. My observation is that almost all printed material concerning microcomputers shares this fault. I live near a very large bookseller and visit it regularly in hopes of finding useful information. This store carries every micro magazine, save 68 Journal, of which I have ever heard, as well as numerous books on the subjects of soft- and hard-ware. Incidentally, 68 J. is the only magazine to which I subscribe although I do leaf through about twenty each month.

Since I have worked in electronics since the early 1950's and have been into microcomputing since 1975 perhaps I could be permitted a few comments on the state of the microcomputing world.

It seems to me that the publishing industry has perfected the alchemical miracle of our era, and one previously in the domain of governments, which is the turning of paper into money via the application of ink. Evidently the main purpose of most magazines is to sell advertising and eventually, products, rather than being committed to the dissemination of useful information. Most of the publications coming into my view contain nothing of value, and the balance contain little. Even the advertisements seldom describe items adequately and some of them lie outright. Perhaps this is more a comment on the present day computerist rather than the manufacturers and publishers as, looking through my collection of old magazines I find a much more considerate approach. I think that somewhen after 1978-79 the industry graduated from a enthusiast viewpoint to a commercial, dollar driven approach to the field.

A similar transition took place in the post WW2 days of Ham Radio. With the release of much service related equipment at surplus prices tremendous interest was generated and magazines were full of articles on conversion and use of such equipment as well as scratch-building (hacking). This period of interest was very important to the progress of electronics in the U.S. as many of us, myself included, came into commercial electronics through hamming. Within a few years, however, there was a rush of "tailor-made" equipment to the market, and this period marked a distinct change in the magazines. Advertisements for factory equipment began to crowd out articles, and the articles changed to describing uses for equipment rather than construction and modification. I regard this as the direct precursor to the Citizens Band craze which reduced an interesting, educational hobby to a pursuit for leather lunged Neanderthals. The result was that many of the manufacturers went out of business and many of the hobbyists did likewise. Today Ham Radio is an empty desert, and this source of enthusiastic amateurs has been lost.

It may be possible to derive a parallel pertinent to the microcomputing field. If only those directly dollar-driven are to create equipment and programs for computing then the result would seem to be monopolization of information because of its value. While I do not decry computer games, especially of the simulation type, I feel that serious progress in the field can only be assured when amateurs are encouraged to realize the creatures of their own imaginations.

The secrecy surrounding hardware documentation, operating systems, and application programs constitutes an almost impassable block for many hobby computerists. It may be arguable that disassembling an operating system is a salutary educational experience. As one who has completed such a task I can assure you that the techniques can be acquired in easier ways. Tracing circuit boards may be the only way to accumulate sufficient knowledge to assist one in building a "dream" interface but much gratitude is due the manufacturer who supplies such information without any demur or indignant suspicion. Worst of all is see a manufacturer who sells unreadable Xerox copies of incorrect, unusable data.

If the future of microcomputing is to follow the history of the once proud automobile industry we will soon have three or four major manufacturers left, and we will all be the poorer for it. I believe that the neglect of the 6809 and now the 68000 is directly due to the preponderance of useful data about processors that are obviously technically inferior.

I must thank those, such as yourself and TSC, who understand this need and attempt to fill it. I can only encourage you to extend your efforts in this direction by your reviews and criticisms of equipment and programs in the 68XX field and as helping to maintain a high standard of excellence in hardware and software.

Sincerely yours,

M. J. Kreinik

M. J. Kreinik

P.S. I am looking forward with great interest to your review of the Digital Systems "Uniboard". It is not well supported by the manufacturer, and deserves some notice as a serious alternative to the Color Computer.

M. J. K.

84a Peach Road,
Auckland,10
New Zealand
Phone 64-9-4447963

68 Micro Journal
P.O. Box 849
Hixson, TN 37343

Dear Sirs,

Recently I completed the design of a 256k dynamic Ram board and thought that, since there aren't many hardware projects in your publication, it may be suitable for publication. If you so desire I could arrange to send you a completed one for evaluation.

Details: the board is designed to look like 4x64k blocks.Rather than make the logic design difficult it was decided not to make the board look like 8x32k blocks or 16x16k blocks. It can start on any 64k boundary within the 1Mbyte addressing range of any 6809 system with mapping ram. The only weird bit is the delay line which is part number TTLDM 75, available from Engineered Components Company, 3580 Sacramento Drive, P.O. Box T, San Luis Obispo, CA 93406. The phone numbers are (805) 544-3800 or (800) 235-4144. The price was around $21-$22 the last I heard. Boards are available from me at the above address for $65 U.S including shipping. I only sell boards,not built up units as this project was done for the local 6809 community and I can't afford to build them up. However the boards are available to those who want to build up their own.

The circuit is enclosed and the set up details are as follows.

1) S1-S4 set the barring. All open bars F000-FFFF. S4 closed bars
8000-FFFF

2) S5-S8 set the 64k start,depending on the setting of the 256k start.
All closed, start at 00000,40000,80000,C0000
All open, start at 10000,50000,90000,D0000
S8 closed,start at 20000,60000,A0000,E0000
S7 closed,start at 30000,70000,B0000,F0000

3) Links A-D set the 256k start 00000,40000,80000,C0000 respectively

The power is arranged by conventional means. Each bank of 64k is powered by a 7805 with decoupling capacitors of 100nF per memory chip and 10uF bulk decoupling. The logic is powered by either an LM 309 or LM323 with plenty of 100nF decoupling capacitors.

People using 1MHz systems can use LS series TTL for the logic. Those with 2MHz systems must use either S series or F series TTL so that the set up time required by the 6809 is fulfilled.

At the moment there are 20 of these boards in use in New

Zealand, so I believe the design is reliable. If anyone does want to buy one the boards are ordered as orders come in so it may take some weeks to fulfil orders. Payment must be either by check,cash or any other way except credit cards.

The circuit is enclosed for you to publish if you so desire.

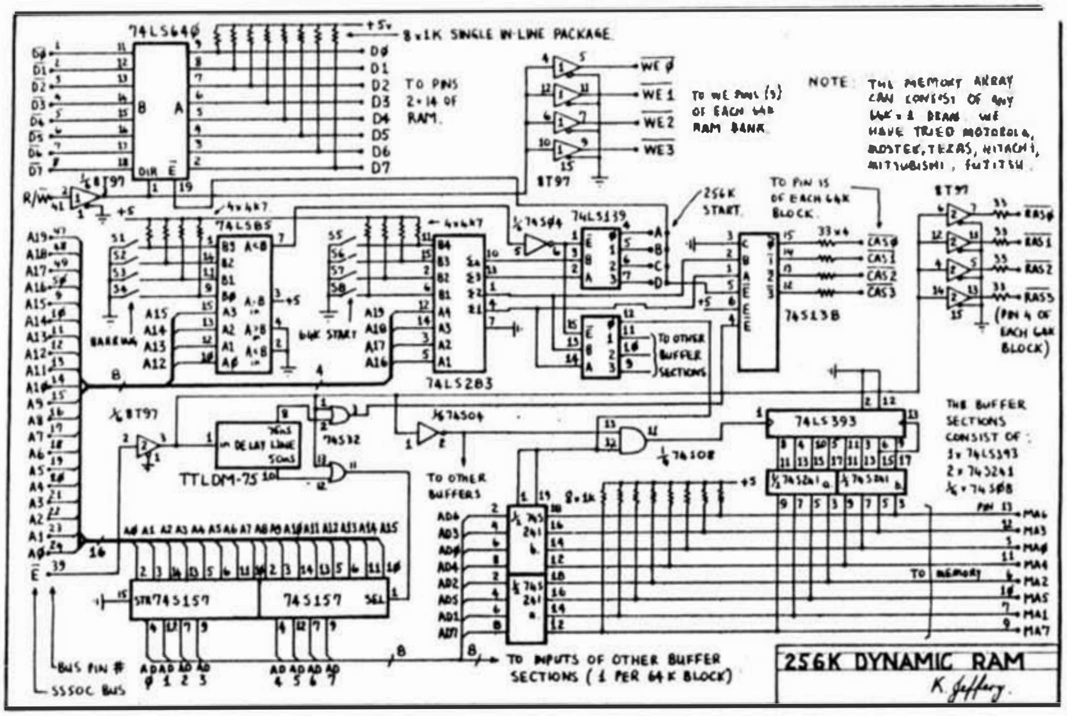
I consider your magazine excellent value for money with more useful articles than many other magazines. Long may you keep up the good work. Comments on 68000-- this computer doesn't really fit in home computers , at least in this country, because of cost and lack of an extensive software base. However I anticipate that this will change as silicon and hopefully, software costs fall. Until then I feel that the level of support you give to the 68000 is adequate.

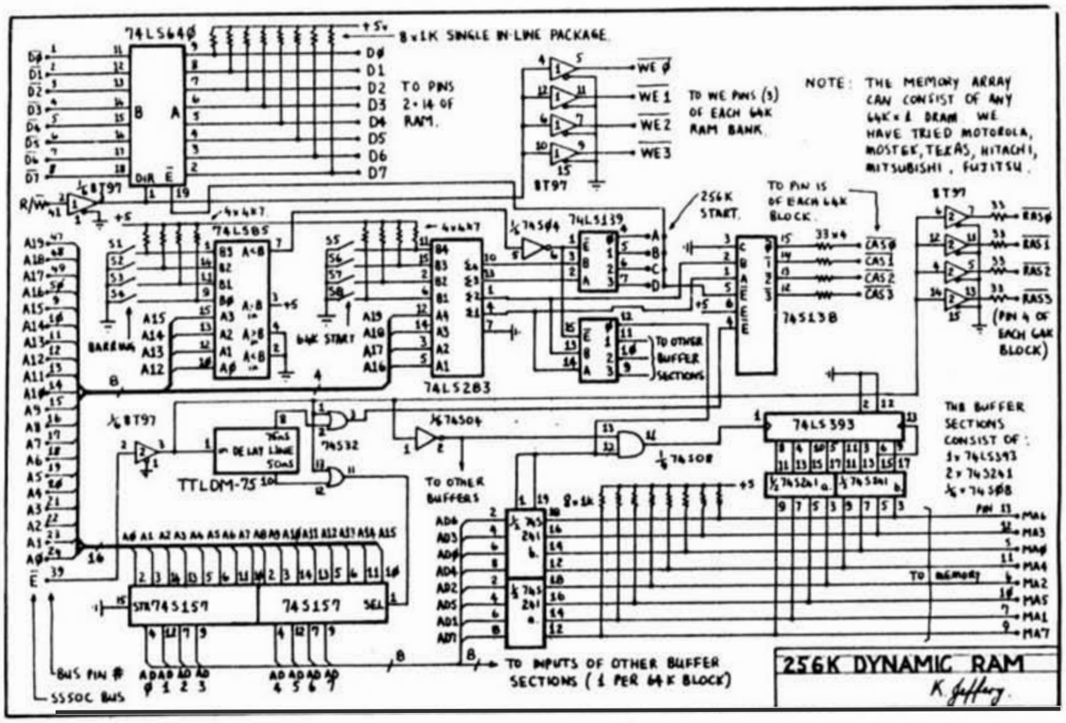
Wishing you luck in the future,

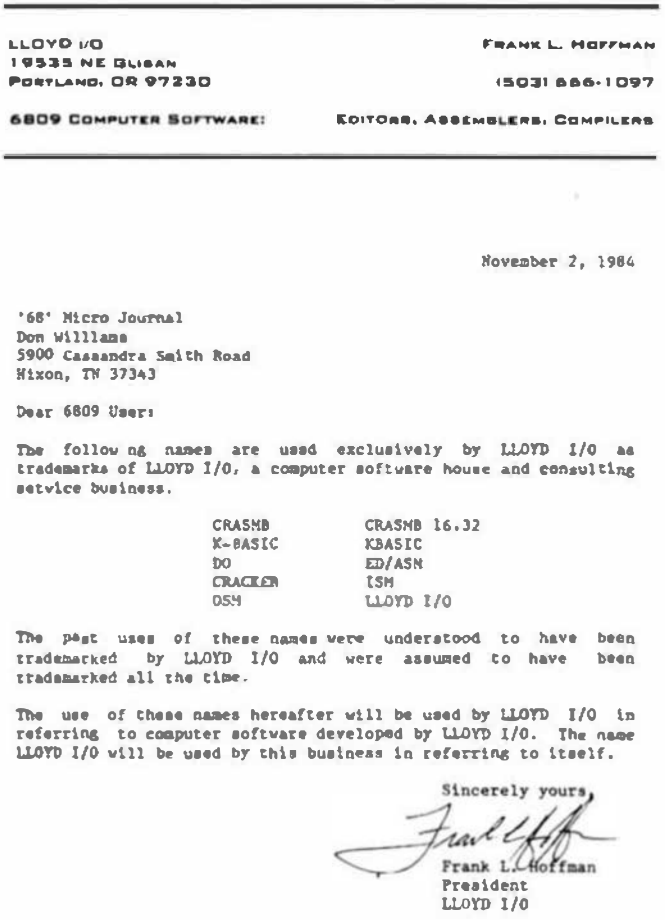
Regards

K. Jeffery.

Keith Jeffery.


74LS640
+5v
8 x 1K SINGLE IN-LINE PACKAGE
D0 D1 D2 D3 D4 D5 D6 D7
TO PINS 2 + 14 OF RAM.
B A
DIR E
R/W
8T97
WE 0
WE 1
WE 2
WE 3
TO WE PINS (3) OF EACH 64K RAM BANK
NOTE: THE MEMORY ARRAY CAN CONSIST OF ANY 64K x 1 DRAM. WE HAVE TRIED MOTOROLA, MOSTEK, TEXAS, HITACHI, MITSUBISHI, FUJITSU.
4 x 4K7
74LS85
S1 S2 S3 S4
BANK
A15 A14 A13 A12
S5 S6 S7 S8
64K START
A19 A18 A17 A16
74LS283
1/6 74S04
74LS139
256K START
TO PIN 15 OF EACH 64K BLOCK
33 x 4
CAS0
CAS1
CAS2
CAS3
74S138
RAS0
RAS1
RAS2
RAS3
(PIN 4 OF EACH 64K BLOCK)
TO OTHER BUFFER SECTIONS
1/4 8T97
DELAY LINE 50nS
TTLDM-75
74S32
1/6 74S04
TO OTHER BUFFERS
1/4 74S08
74LS393
THE BUFFER SECTIONS CONSIST OF: 1 x 74LS393 2 x 74S241 1/4 x 74S08
8 x 1K
74S241
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
74S157
74S157
SEL
STR
TO MEMORY
MA6 MA3 MA0 MA4 MA2 MA5 MA1 MA7
AD6 AD3 AD0 AD4 AD2 AD5 AD1 AD7
BUS PIN #
SS50C BUS
AD AD AD AD 0 1 2 3
AD AD AD AD 4 5 6 7
TO INPUTS OF OTHER BUFFER SECTIONS (1 PER 64K BLOCK)
256K DYNAMIC RAM
K. Jeffery.





LLOYD I/O
19535 NE GLISAN
PORTLAND, OR 97230

FRANK L. HOFFMAN

(503) 666-1097

6809 COMPUTER SOFTWARE: EDITORS, ASSEMBLERS, COMPILERS

November 2, 1984

'68' Micro Journal
Don Williams
5900 Cassandra Smith Road
Hixon, TN 37343

Dear 6809 User:

The follow ng names are used exclusively by LLOYD I/O as trademarks of LLOYD I/O, a computer software house and consulting setvice business.

CRASMB　　CRASMB 16.32
X-BASIC　　XBASIC
DO　　ED/ASM
CRACKER　　ISM
OSM　　LLOYD I/O

The past uses of these names were understood to have been trademarked by LLOYD I/O and were assumed to have been trademarked all the time.

The use of these names hereafter will be used by LLOYD I/O in referring to computer software developed by LLOYD I/O. The name LLOYD I/O will be used by this business in referring to itself.

Sincerely yours,

Frank L. Hoffman
President
LLOYD I/O

Enclosed is a copy of a command I wrote for my SSB DOS68.51. It is an alternative to SDC.3 (Single Disk Copy) to be used when a number of files and/or commands are to be copied.

The program cycles through all files and commands in the disk directory and asks the user if each should be copied. The program responds to Y or N only; any other response will result in the question being asked again. If the file already exists on the target disk, the user is informed and the process continues. This program is my style of programming (Brute Force) and is not an example of efficiency. However, it works.

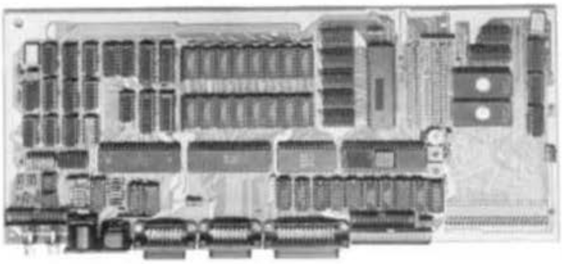
One note: the call is not the same as SDC.3 because 1) no file name is specified and 2) no memory limit can be entered. The memory limit is assumed to be 16K ($4000) but can be changed in line 184.

Sincerely
John [illegible]
245 S.W. Wildwood Ln.
Bremerton Wa. 98310

```
                   1:        NAM   SINGLE DISK CONTINUOUS COPY

                   3: ***********************************************
                   4: *                                             *
                   5: * TRANSIENT COMMAND TO CONTINUOUSLY COPY ON A *
                   6: * SYSTEM WITH ONLY ONE DISK DRIVE             *
                   7: *                                             *
                   8: ***********************************************

7289              10: ZINCH  EQU   $7289
72A9              11: ZTYPDE EQU   $72A9
7783              12: CDFM   EQU   $7783
7283              13: ZWARMS EQU   $7283
7786              14: DFM    EQU   $7786
72A6              15: ZOUTST EQU   $72A6
7286              16: ZOUTCH EQU   $7286

0100              18:        ORG   $0100
0100 CE 02BA      19: START  LDX   #CRLF
0103 BD 72A6      20:        JSR   ZOUTST
0106 7F 0308      21:        CLR   FCB+2        SET UNIT NUMBER = 0
0109 86 0A        22:        LDA A #10          SETUP DIRECTORY READ
010B B7 0306      23:        STA A FCB
010E BD 01EE      24:        JSR   FILOC
0111 7F 026D      25:        CLR   COUNT        COUNT = # OF FCBS READ; INITIALLY ZERO
0114 86 0B        26: LOOP1  LDA A #11          READ FCB
0116 7C 026D      27:        INC   COUNT
0119 B7 0306      28:        STA A FCB
011C BD 01EE      29:        JSR   FILOC
011F CE 0271      30:        LDX   #MSG1        PRINT "FILE:"
0122 BD 72A6      31:        JSR   ZOUTST
0125 BD 021D      32:        JSR   PRTNAM       PRINT FILE NAME
0128 CE 0279      33: LOOP2  LDX   #MSG2        ASK IF USER WISHES TO COPY
012B BD 72A6      34:        JSR   ZOUTST
012E BD 7289      35:        JSR   ZINCH        ANSWER Y OR N
0131 36           36:        PSH A
0132 CE 02BA      37:        LDX   #CRLF        PRINT CARRIAGE RETURN / LINE FEED
0135 BD 72A6      38:        JSR   ZOUTST
0138 32           39:        PUL A
0139 81 4E        40:        CMP A #'N          IF "N", READ NEXT FCB
013B 27 D7        41:        BEQ   LOOP1
013D 81 59        42:        CMP A #'Y
013F 26 E7        43:        BNE   LOOP2        IF NOT "Y", ASK AGAIN
0141 CE 03AC      44:        LDX   #FILE        IF "Y", POINT TO BEGINNING OF STORAGE AREA
0144 86 04        45:        LDA A #4           OPEN FILE FOR READ
0146 B7 0306      46:        STA A FCB
0149 BD 01EE      47:        JSR   FILOC
014C 86 05        48:        LDA A #5           READ FROM FILE
014E B7 0306      49:        STA A FCB
0151 BD 0248      50: READAG JSR   FILRW
0154 7D 026E      51:        TST   FLAG         END OF FILE?
0157 26 13        52:        BNE   REDCLS       YES, BRANCH
0159 BC 026F      53:        CPX   MEMMAX       MEMORY LIMIT REACHED?
015C 27 05        54:        BEQ   MEMERR       YES, BRANCH
015E A7 00        55:        STA A 0,X
0160 08           56:        INX
0161 20 EE        57:        BRA   READAG       IF READ SUCCESSFUL, READ AGAIN
0163 CE 02F9      58: MEMERR LDX   #MSG8        PRINT "MEMORY LIMIT" AND RESTART DOS
0166 BD 72A6      59:        JSR   ZOUTST
0169 7E 0207      60:        JMP   OUT

016C 7F 026E      62: REDCLS CLR   FLAG
016F 09           63:        DEX
0170 FF 026B      64:        STX   LAST         STORE ADDRESS OF LAST DATA BYTE
0173 86 06        65:        LDA A #6           CLOSE FILE
0175 B7 0306      66:        STA A FCB
0178 BD 01EE      67:        JSR   FILOC
017B CE 028F      68:        LDX   #MSG4        TELL USER TO CHANGE DISKETTES
017E BD 72A6      69:        JSR   ZOUTST
0181 BD 7289      70:        JSR   ZINCH
0184 CE 02BA      71:        LDX   #CRLF
0187 BD 72A6      72:        JSR   ZOUTST
018A 86 01        73:        LDA A #1           OPEN FILE FOR WRITE
018C B7 0306      74:        STA A FCB
018F BD 01EE      75:        JSR   FILOC
0192 7D 026E      76:        TST   FLAG         ON RETURN, CHECK FLAG
0195 26 15        77:        BNE   FLGSET       BRANCH IF SET
0197 86 02        78:        LDA A #2           IF NOT, SETUP TO WRITE FILE
0199 B7 0306      79:        STA A FCB
019C CE 03AC      80:        LDX   #FILE
019F A6 00        81: WRTAGN LDA A 0,X          GET BYTE
01A1 BD 0248      82:        JSR   FILRW
01A4 BC 026B      83:        CPX   LAST         END OF DATA?
01A7 27 0F        84:        BEQ   WRTCLS
01A9 08           85:        INX                NO, CONTINUE
01AA 20 F3        86:        BRA   WRTAGN
01AC 7F 026E      87: FLGSET CLR   FLAG         RESET FLAG
01AF CE [illegible] 88:      LDX   #MSG7        TELL USER FILE ALREADY EXISTS
01B2 BD 72A6      89:        JSR   ZOUTST
01B5 7E 01C6      90:        JMP   RESTRT
01B8 CE 02AD      91: WRTCLS LDX   #MSG5        PRINT "FILE COPIED"
01BB BD 72A6      92:        JSR   ZOUTST
01BE 86 03        93:        LDA A #3           CLOSE FILE
01C0 B7 0306      94:        STA A FCB
01C3 BD 01EE      95:        JSR   FILOC
01C6 CE 02BD      96: RESTRT LDX   #MSG6        TELL USER TO REINSERT SOURCE DISK
01C9 BD 72A6      97:        JSR   ZOUTST
01CC BD 7289      98:        JSR   ZINCH
01CF CE 02BA      99:        LDX   #CRLF
01D2 BD 72A6     100:        JSR   ZOUTST
01D5 F6 026D     101:        LDA B COUNT        GET # OF FCBS ALREADY READ
01D8 86 0A       102:        LDA A #10          SETUP DIRECTORY READ
01DA B7 0306     103:        STA A FCB
01DD BD 01EE     104:        JSR   FILOC
01E0 86 0B       105:        LDA A #11          READ FIRST FCB
01E2 B7 0306     106:        STA A FCB
01E5 BD 01EE     107: LOOP3  JSR   FILOC
01E8 5A          108:        DEC B
01E9 26 FA       109:        BNE   LOOP3        LOOP THROUGH FCBS ALREADY READ
01EB 7E 0114     110:        JMP   LOOP1

                 112: ***********************************************
                 113: *  SUBROUTINE TO OPEN AND CLOSE FILES         *
                 114: ***********************************************

0    FF 0269      16: FILOC  STX   SAVE         SAVE INDEX REG
01F1 36          117:        PSH A
01F2 CE 0306     118:        LDX   #FCB         LOAD ADDRESS OF FCB
01F5 BD 7786     119:        JSR   DFM          DFM I/O REQUEST
01F8 27 16       120:        BEQ   OK           RETURN IF NO ERROR
01FA A6 01       121:        LDA A 1,X          GET ERROR STATUS CODE
01FC 81 06       122:        CMP A #6           CHECK FOR END OF DIRECTORY
01FE 27 15       123:        BEQ   END
0200 81 02       124:        CMP A #2           CHECK FOR "FILE ALREADY EXISTS" ERROR
0202 27 09       125:        BEQ   EXISTS
0204 BD 72A9     126:        JSR   ZTYPDE       PRINT ERROR CODE,CLOSE FILES,RESTART DOS
0207 BD 7783     127: OUT    JSR   CDFM
020A 7E 7283     128:        JMP   ZWARMS
020D 7C 026E     129: EXISTS INC   FLAG         SET FLAG IF FILE ALREADY EXISTS
0210 FE 0269     130: OK     LDX   SAVE
0213 32          131:        PUL A
0214 39          132:        RTS
0215 CE 0281     133: END    LDX   #MSG3        PRINT "END DIRECTORY" AND EXIT
0218 BD 72A6     134:        JSR   ZOUTST
021B 20 EA       135:        BRA   OUT

                 137: ***********************************************
                 138: *  SUBROUTINE TO PRINT FILE NAME              *
                 139: ***********************************************

021D FF 0269     141: PRTNAM STX   SAVE         SAVE INDEX REG
0220 C6 06       142:        LDA B #6           PRINT FIRST 6 CHARACTERS
0222 CE 0309     143:        LDX   #FCB+3
0225 A6 00       144: PRLOP1 LDA A 0,X
0227 BD 7286     145:        JSR   ZOUTCH
022A 08          146:        INX
 22B 5A          147:        DEC B
022C 26 F7       148:        BNE   PRLOP1
022E 86 2E       149:        LDA A #'.          PRINT DECIMAL PT
0230 BD 7286     150:        JSR   ZOUTCH
```

```
0233 C6 03                   LDA B #3        PRINT EXTENSION
0235 CE 030F                 LDX   #FCB+9
0238 A6 00         PRLOP2    LDA A 0,X
023A BD 7286                 JSR   ZOUTEE
023D 08                      INX
023E 5A                      DEC B
023F 26 F7                   BNE   PRLOP2
0241 CE 02BA                 LDX   #CRLF
0244 BD 72A6                 JSR   ZOUTST
0247 39                      RTS

                   ***********************************************
                   * SUBROUTINE TO READ AND WRITE FILE           *
                   ***********************************************

0248 FF 0269       FILRW     STX   SAVX       SAVE INDEX REG
024B CE 0306                 LDX   #FCB       LOAD ADDRESS OF FCB
024E BD 7786                 JSR   DFM        DFM I/O REQUEST
0251 27 12                   BEQ   RWCK       RETURN IF NO ERROR
0253 A6 01                   LDA A 1,X        CHECK FOR END OF FILE
0255 81 08                   CMP A #8
0257 27 09                   BEQ   ENDRD
0259 BD 72A9                 JSR   ZTYPIE
025C BD 7783                 JSR   CDFM
025F 7E 7283                 JMP   ZWARMS
0262 7C 026E       ENDRD     INC   FLAG
0265 FE 0269       RWCK      LDX   SAVX
0268 39                      RTS

0269 00 00         SAVX      FDB   0000
026B 00 00         LAST      FDB   0000
026D 00            COUNT     FCB   0
026E 00            FLAG      FCB   0
026F 40 00         MEMAX     FDB   $4000      16K MEMORY ASSUMED
0271 20            MSG1      FCC   / FILE: /
0278 00                      FCB   0
0279 20            MSG2      FCC   / COPY? /
0280 00                      FCB   0
0281 45            MSG3      FCC   /END DIRECTORY/
028E 00                      FCB   0
028F 43            MSG4      FCC   /CHANGE DISKS AND HIT ANY KEY /
02AC 00                      FCB   0
02AD 20            MSG5      FCC   / FILE COPIED /
02BA 0A            CRLF      FCB   $0A,$0D,0
02BD 52            MSG6      FCC   /REMOUNT SOURCE DISK AND HIT ANY KEY /
02E1 00                      FCB   0
02E2 46            MSG7      FCC   /FILE ALREADY EXISTS /
02F6 0A                      FCB   $0A,$0D,0
02F9 4D            MSG8      FCC   /MEMORY LIMIT/
0305 00                      FCB   0
0306               FCB       RMB   166
03AC               FILE      RMB   1
0100                         END   START
     NO ERROR(S) DETECTED
```

```
  SYMBOL TABLE:
CDFM     7783     COUNT    026D     CRLF     02BA     DFM      7786
END      0215     ENDRD    0262     EXISTS   0200     FCB      0306
FILE     03AC     FILOC    01EE     FILRW    0248     FLAG     026E
FLRSET   01AC     LAST     026B     LOOP1    0114     LOOP2    0128
LOOP3    01E5     MEMAX    026F     MEMERR   0163     MSG1     0271
MSG2     0279     MSG3     0281     MSG4     028F     MSG5     02AD
MSG6     02BD     MSG7     02E2     MSG8     02F9     OK       0210
OUT      0207     PRTRAN   021D     PRLOP1   0225     PRLOP2   0238
REDADR   0151     REDCLS   016C     RESTRT   0106     RWCK     0265
SAVX     0269     START    0100     WRTADR   0197     WRTCLS   0180
ZINCH    7289     ZOUTEE   7286     ZOUTST   72A6     ZTYPIE   72A9
ZWARMS   7283
  FILE: DFM680.352
  COPY? N
  FILE: DFM680.353
  COPY? N
  FILE: LIST.$
  COPY? Y
 CHANGE DISKS AND HIT ANY KEY
 FILE ALREADY EXISTS
 REMOUNT SOURCE DISK AND HIT ANY KEY
  FILE: SOCC3.BAK
  COPY? Y
 CHANGE DISKS AND HIT ANY KEY
 FILE ALREADY EXISTS
 REMOUNT SOURCE DISK AND HIT ANY KEY
  FILE: DELETE.$
  COPY? N
  FILE: SOCC3.BIN
  COPY? N
  FILE: RENAME.$
  COPY? Y
 CHANGE DISKS AND HIT ANY KEY
  FILE COPIED
 REMOUNT SOURCE DISK AND HIT ANY KEY
  FILE: SOCC.TXT
  COPY? N
  FILE: EDIT.$
  COPY? N
  FILE: ASMB.$
  COPY? Y
 CHANGE DISKS AND HIT ANY KEY
  FILE COPIED
 REMOUNT SOURCE DISK AND HIT ANY KEY
  FILE: SOCC.BAK
  COPY? Q
  COPY? N
  FILE: SOCC.$
  COPY? N
  FILE: SAVE.$
  COPY? N
  FILE: SIZE.$
  COPY? N
 END DIRECTORY
```



Classified Advertising

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. **LIKE NEW** - New cost $1295.00 - ONLY **$759.00** ready to run - Call Tom - Larry - Bob, CPI 615 842-4600

***

For Sale: Motorola 128K Memory Boards, removed from SWPTC S/09 $795.00, SWPTC 8212 Terminals Demostrators $795.00, Hazelwood Dynamic 6 K Memory Boards $395.00 Call ask for Tom 615/ 842-4600

***

SWTPC 6809 S-09 COMPUTER SYSTEM with 20 meg hard disk, dual 8" drives, 2 serial cards, and 128k SWT memory. This computer is set up to run UniFLEX and has been used as backup system. Memory needs updating. As removed from service. A great buy for those wanting the power of a UNIX-like system. $4,950.00
Call 1-800-255-1382 Ext. 47.

***

FOR SALE: PR-40 printer $60, APTEK 4K 1702 SS50 EPROM board, F&D 1702 programmer (uses MPLA) both $100, Universal Data 212/103 modem $300, DS68 103 modem card populated, never used, with EDC coupler $50.
Gordon (504)889-1224

***

GIMIX OS9 LII 6809 System. GIMIX 6809 "PLUS" CPU, GIMIX DMAF2 8/5" DMA disk controller, 192K dynamic RAM, two 8" DSDD QUME drives, F&D video, cherry keyboard, 4 I/O Ports, Motorola monitor. Very Reliable. Complete documentation. Also Base 2 Printer $175.
Call (312) 382-5478 after 7 PM.

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishing Inc.,
Is offering the following "SUBSCRIBER SERVICE":

COMPILER COMPARISION AND EVALUATION REPORT

Due to the constant and rapid updating and enhanchment of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the verious claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

PASCAL "C" GSPL WHIMISCAL PL/9

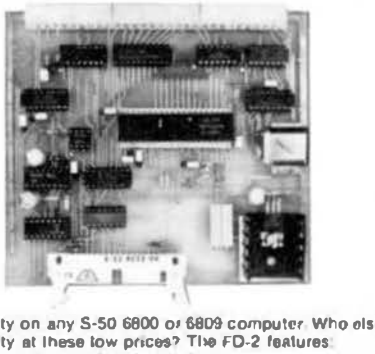
Initial Subs ription - $39.95
(Includes 1 year updates)
Updates for 1 year - $14.50

S.E. MEDIA - CPI
5900 Cassandra Smith, POB 794
Hixson, TN 37343
615 842-4601

NEW

OS-9 SUPPORT FOR FD-2

NEW!



NEW!

Run double density on any S-50 6800 or 6809 computer. Who else can offer this capability at these low prices? The FD-2 features:

- Control of up to four 5¼" DS/DD Drives
- SS-30 or SS-30C compatible
- Use Flex, OS-9, or Star Dos operating systems
- 2.0 MHZ operation with no "slow I/O" required
- Compatible with SWTPC DC1, DC2, DC3, or DC4 controllers

FD-2	Assembled Tested Controller Card	$149.95
DRV-68	6800 double density drivers + format program	$ 19.95
DRV-69	6809 double density drivers - format program	$ 29.95
DRV-09	FD-2 Disk Drivers for OS-9 (Source)	$100.00
STAR-DOS	For SWTPC & FD-2	$ 75.00

PERIPHERAL TECHNOLOGY
"Supplying Floppy Disk Controllers Since 1978"
3760 Lower Roswell Road
Marietta, Georgia 30067
VISA/MASTERCARD/CHECK/COD 404/973-0042

"OS-9 is a trademark of Microware and Motorola.

K-BASIC
for OS9 & FLEX
$199

K-BASIC is a complete BASIC compiler package including: the compiler itself, the assembler, documentation, and sample programs. It features six atomic data types including: real numbers; strings; 8 bit, 16 bit, 32 bit, and 64 bit signed integers. All types may be dimensioned with one or two subscripts. K-BASIC converts programs to MACHINE language code which may be put into EPROMS or ROMS.

K-BASIC syntax is very close to TSC's BASIC and XBASIC interpreters. Line numbers are not required (may be up to 16 characters). Variable names may be up to 12 characters long. The AT statement dimensions variables to absolute memory addresses.

The future of K-BASIC will see additional versions for the assorted interpreters currently available. This means you can compile your BASIC programs you now have.

Call (503) 666-1097 for our CATALOG, we have many other programs including: DO . . . $69 OSM . . . $99 ED/ASM . . . $69

CRASMB
for OS9 & FLEX
$399

CRASMB is the highly acclaimed cross assembler package for OS9 and FLEX systems, and is the only one of its type available. It turns your computer into a development station for these CPUs:

6800 6801 6804 6805 6809 6811 6502
7000 1802 8048 8051 8080 8085 Z80
(68000 16/32 bit cross assembler . . . $249)

CRASMB features include: Macros, Conditional assembly, Library file calls (12 deep), Symbol length to 30 characters, Symbol cross reference tables, Object code in 4 formats (OS9, FLEX, S1-S9, INTEL HEX), plus many other extended directives and options not found on other assemblers.

LLOYD I/O 19535 NE GLISAN, PORTLAND, OR 97230 USA
Phone: (503) 666-1097 (Software Consultation Available)

VISA, MC, COD, CHECK, APPROVED P.O.'s ACCEPTED

England: Vivaway (0582 423425), Windrush (0692 405189)
Germany: Zacher Computer (65 25 299)
Australia: Palls Radio Electronics (61 2 344 9111)

OS9 is a ™ of Microware, FLEX is a ™ of TSC

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL
CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

FREE DISKETTE WITH EVERY $50 PURCHASE

☎ TOLL FREE
1-800-338-6800
For Ordering

TELEX 558 414 PVT BTH

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

!!! SPECIALS !!!

"Year End CLEARANCE" --- While they Last

FLEX Software ---------

TSC "FLEX Utilities"	was $75.00,	NOW only	$60.00
TSC "Sort Merge"	was $75.00,	NOW only	$ .00
TSC "6809 BASIC"	was $75.00,	NOW only	$60.00
TSC "Extended BASIC"	was $100.00,	NOW only	$85.00
TSC "DeBug"	was $75.00,	NOW only	$60.00
TSC "FLEX Diagnostics"	was $75.00,	NOW only	$60.00
TSC "Text Processing System"	was $75.00,	NOW only	$60.00
TSC "68000 Cross Assembler"	was $250.00,	NOW only	$220.00

LUCIDATA "TEKPAK" (FLEX9, 5 1/4") -- A Pascal implementation of the Tektronix 40xx Terminal Control System, with Pascal SOURCE. The Manual includes a discussion of how to utilize this package in the graphical library in implementing Vector Drawing, Point Plotting, etc.. up through Windowing and Clipping concepts.
Normal Price, $100.00: NOW only $85.00

OS-9 Software ------

MICROWARE "OS-9 File Handler Toolbox":	NOW only	$70.00
MICROWARE "Relocating Macro Assembler":	NOW only	$170.00

!!! NEW NEW NEW !!!

Computer Systems Consultants, Inc.

CRODEM TELECOMMUNICATIONS PROGRAM
Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "MODEM7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, CCF, OS-9, UniFLEX: with complete Source - $100.00
without Source - $50.00

X-E-M -- TRUE (not Macros) CROSS ASSEMBLERS
Use your 6809-Based Computer System for developing Software for 1802/5, 6800/01/03/11, 6804, 6805, 6809, 6502/3, 8080/5, 8048, 8051, Z-80, and 68000 Systems. Provides the Assembler Language and Listings normally used on the target Systems. Written in "C"; produces Motorola S-Text for machine independence.

FLEX, CCF, OS-9, UniFLEX
each - $60.00
any 3 - $100.00
the complete set (including the C Source) - $200.00

LLOYD I/O

K-BASIC -- A "Native Code" BASIC **Compile**
Level I K-BASIC supports sequential files, floating point, 3 sizes of integers, string variables, and arrays. The single-pass compiler compiles to Assembly Language Source Code (which may be assembled by the included OSM Assembler, or by the CRASMB Cross Assemblers). Conditional assembly is used to reduce the size of the run time package. (See Review in Oct. '84 Issue of '68' Micro Journal.)

FLEX, CCF, OS-9 Compiler with OSM Assembler - $199.00

OSM -- Extended 6809 Macro Assembler
Provides local labels, Motorola S-records, and Intel Hex records. Also generates OS-9 Memory modules under FLEX, allowing the maintenance of source code programs for both DOS's on one System.

FLEX, CCF, OS-9 **$99.00**

CRASMB -- 8-Bit Macro Cross Assembler
Same features as OSM, cross-assembles to 6800/2/8, 6801/3, 6804, 6805, 6809, 6811, 6502, 1802, 8048, 8080/5, Z-8, Z-80. Fully supports the target chip's standard mnemonics and addressing modes.

FLEX, CCF, OS-9 full package -- **$399.00**

CRASMB 16.32 -- Cross Assembler for the **68000**
Same features as 8-Bit Cross Assemblers above
$249.00

Compusense Ltd.

CRUNCH COBOL -- COBOL Compiler
This COBOL Compiler supports a large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full support of the FLEX File Structures is implemented including Random Files and the ability to process Keyed Files. Large programs can be segmented and linked at runtime, or imolemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.

FLEX, CCF Normally $199.00
Special Introductory Price (while they last) -- **$99.95**

ASSEMBLERS

Southeast Media

ASTRUK09

A "Structured Assembler for the 6809" which requires the TSC **Macro Assembler**. Allows direct use of structured statements such as IF, ELSE, DO, REPEAT, etc., and provides indented level formatting of the listing so that the structure is apparent. Re. '68' **Micro Journal**, Sept. '83 (program was called "STASM09"; has been renamed due to conflicts).

A User reports

"... I'm very pleased and am now writing almost exclusively in (ASTRUK09). I've selected it over --- for all future systems development... As (one) of my early evaluations, I rewrote a rather elaborate routine originally done in assembly. Out of the 1000 bytes of code generated, the (ASTRUK09) version used only 20 more bytes than the original. --- could not handle this program since it uses triple-precision fixed point arithmetic... I have a large body of code already written that is incompatible with --- constructs. No oroblem with (ASTRUK09) and the structure sure helps in understanding the logic!"

F. CCF - $99.95

TSC

Macro Assembler

The FLEX STANDARD Assembler. Special -- F,CCF $35.00

Relocating Assembler w/Linking Loader

Use with many of the C and Pascal Compilers. F,CCF $150.00

Great Plains Comp. Co.

RMMAC

Relocating, Recursive Macro Assembler and Linking Loader.
F,CCF $120.00; w/Source $240.00

OmegaSoft

PRALLI

Relocating Assembler and Linking Loader
F,CCF $125.00; for One Year Maint., add $50.00

Windrush Micro Systems
MACE, by Graham Trott. F,CCF - $98.00

VISA

MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

☎ TOLL FREE
1-800-338-6800
For Ordering
SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends ---
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

DISASSEMBLERS

Computer Systems Consultants

SUPER SLEUTH

Computer Systems Consultants Super Sleuth is a "Time Tested", reliable, PROVEN Disassembler that has gained acceptance through out the SS-50 Bus Community as an extremely POWERFUL, INTERACTIVE, Software Tool. The Super Sleuth Software Package consists of 3 Programs; SLEUTH (the Disassembler), CHGNAM (used to globally Change Labels to a meaningful Name), and XREF (a Cross Reference Generator for Source Code Files). SLEUTH will Disassemble Memory Resident 6809 Code and 6800, 6801, 6802, 6803 (the "Baby CoCo"), 6805, 6808, 6809, and 6502 (Apple, Atari, Commodore, etc.) Binary Disk Files. (See Aug. '83 '68' Micro Journal "Color Users Notes" Column for a full Review.)

Color Computer	SS-50 Bus (all w/ Source)
CCO (32K Req'd)	
Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00	

ALL Computer Systems Consultants Software
runs on the Color FLEX Systems
ALL in stock
call 800-338-6800
for IMMEDIATE DELIVERY

Computer Systems Center

DYNAMITE +

An "easy to use", powerful Disassembler for Disk Resident 6809 and 6800 Binary Files. Allows the development of a "Control File" of various Program "Boundaries" during successive disassemblies; can use a Label File which automatically replaces a Hex Location with a Label Name; Includes an XREF Utility; etc. Label Files provided for Mini-FLEX, FLEX2, FLEX9, Color Computer (for use with Color FLEX Systems), etc. OS-9 Version includes special OS-9 options.

CCF, Obj. Only $100.00	CCO, " " $150.00
F, " " $100.00	O, " " $150.00
U, " " $300.00	

COMPILERS and DECOMPILERS

6809 "Structured" Assembly Lang. Compilers

Windrush Micro Systems

PL/9

By Graham Trott. A combination Editor/Compiler/Debugger, all in ONE PACKAGE; provides a totally INTERACTIVE Program Development Cycle. The Single-Pass Compiler supports large Symbol Names; Variable Types; Pointers; Control Structures (similar to 'C' or 'Pascal'); Stack, A- ,B-, and D-Register manipulation; etc. The Source-Oriented Trace/Debugger provides Single Stepping, Breakpointing, etc. An excellent Software Development Tool which provides for the maximum utilization of the power of the 6809.

F, CCF - $198.00

Whimsical Developments

WHIMSICAL

Need the Ease of Design and Maintainability of "Structured Programming" AND the Speed and Control of Assembly Language? Then WHIMSICAL was designed for you! This Single Pass, Recursive Descent Compiler provides the tool for developing simple Utilities to MAJOR Systems in Assembly Language. Supports 3 "Lex" Levels which allow one level of Procedure nesting, or more within "Modules". It is easy to develop programs written for other machines since you are working at the Assembly Language level. Features unified, user-defined I/O; produces ROMable, relocatable, recursive, re-entrant Code; Structured style and statements with Procedures and Modules; supports Byte and Double-Byte primitives with 3 types of Integers (up to 32 bit), Char and Boolean, and unlimited sized Arrays (vectors only); Interrupt handling; unlimited length Variable Names; Variable Initialization (defaults to $00); Include "Source File" directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. To quote Ron Anderson in his comments about WHIMSICAL in the Sept. '83 Issue of '68' Micro Journal that, except for the lack of floats, "..... **I have to give this one VERY high rating, ...**". It is a FAST Compiler which produces FAST Code (his "Primes" Benchmark ran at 9 secs. on a 2 MHz System).

F and CCF - $195.00

FREE DISKETTE WITH EVERY $50 PURCHASE

☎ TOLL FREE
1-800-338-6800
For Ordering

TELEX 558 414 PVT BTH

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for Information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

'C' COMPILERS

Windrush Micro Systems

C Compiler

By James McCosh. Full featured C Compiler for the FLEX Operating System (lacking ONLY "bit-fields"), including an Assembler. Requires the TSC Relocating Assembler IF the user wishes to implement his own Libraries.

F and CCF - $295.00

Introl

C Compiler

A full-featured C, streamlined for the 6809. Generates very efficient object code. Output "benchmarks" close to 10MHz 68000 in 8 Bit Operations; 1.5 times faster than a 4 MHz Z80 when using a 2MHz 6809 System (Re. p 43, "68" Micro Journal, May '83). Floats, etc.

F, CCF, and O	$375.00
U	$425.00
One Year Maint.	$100.00

PASCAL COMPILERS

TSC

PASCAL Compiler

Native Code Compiler (UCSD Oriented).

F and CCF - $200.00

Lucidata

PASCAL Compiler

P-Code Compiler (ISO Standard). Designed especially for Microcomputer Systems; Run-time System checks available resources for each task, allowing operation on even minimal computer systems. Allows linkage to Assembler Code for maximum flexibility.

F and CCF 5"	$190.00
F 8"	$206.00

OmegaSoft

PASCAL Compiler

For the PROFESSIONAL; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Use custom I/O devices in place of the Pascal INPUT and OUTPUT; Long Int. (32 Bit); Dynamic length strings; Interrupt processing, ROM-able, PIC, Re-Entrant Code, etc. POWERFUL! Includes Source for the Symbolic Debugger, Runtime, and several Utilities. Requires a "Motorola Compatible" Relocating Assembler and Linking Loader.

F and CCF	$425.00
One Year Maint.	$100.00

DECOMPILERS

Southeast Media

DUB (A UniFLEX "basic" De-Compiler)

Re-Create a Source Listing from UniFLEX Compiled basic Programs. Easy to Use; works w/ ALL Versions of UniFLEX basic; Output to Disk or Terminal. Time TESTED and PROVEN; SOLID!

U - $219.95

VISA

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

☎ TOLL FREE
1-800-338-6800
For Ordering

SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
Info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —

F = FLEX, CCF = Color Computer FLEX	
O = OS-9, CCO = Color Computer OS-9	
U = UniFLEX	
CCD = Color Computer Disk	
CCT = Color Computer Tape	

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

FREE DISKETTE WITH EVERY $50 PURCHASE

☎ TOLL FREE
1-800-338-6800
For Ordering

TELEX 558 414 PVT BTH

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

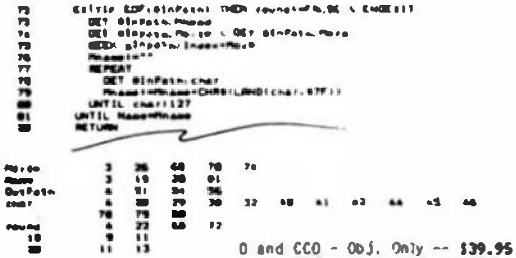
CoCo OS-9™ FLEX™
SOFTWARE

UTILITIES

Southeast Media

Basic09 XRef

This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also included is a Program List Utility which outputs the listing without the overhead of building the cross reference table, which allows it to run considerably faster when only a "pretty printed" listing with line numbers is desired. Requires Basic09 or RunB for operation.



```
72        [illegible]
73          GET #inPath,[illegible]
74          GET [illegible] GET #inPath,[illegible]
75          [illegible]
76          [illegible]
77          REPEAT
78            GET #inPath,char
79            [illegible]=CHR$(LAND(char,$7F))
80          UNTIL char>127
81        UNTIL [illegible]
82        RETURN

[illegible]      3   26   68   78   79
[illegible]      3   19   38   81
[illegible]      4   11   34   56
[illegible]      4   28   29   30   32   40   41   43   44   45   46
                 78   79   80
[illegible]      4   22   68   72
   10            9   11
   [illegible]  11   13
```

O and CCO - Obj. Only -- $39.95
O and CCO - w/ Source - $79.95

Southeast Media

O-F

---- OS/9 to FLEX - FLEX to OS/9 ----

Finally; the barrier has been removed between OS/9 and FLEX formatted disks! Now you can READ **from** and WRITE **to** a Single Sided 5" or 8" FLEX diskette from OS-9 with O-F. O-F is a new and unique program, written in BASIC09 (with Source), that performs the following functions;

REFORMAT: A BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX.

FLEX: A BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk, all selected from a user-friendly menu. Functions provided include reading the FLEX Directory, Deleting FLEX Files, Copying both directions, etc. All selections are interactive and complete, including all necessary prompts to the operator.

FLEX users can read, write and use the special disk as any other FLEX disk, provided the FLEX directory is not allowed to continue beyond track zero (too many files).

O - $79.95

Southeast Media

COPYMULT

--- Copy **LARGE** Disks to several **smaller** disks ---

The following FLEX utilities allow the backup of **ANY** size disk to any SMALLER size diskettes (Winchester to 8's or 5's, 8" to 5's, etc.). By simply inserting diskettes as requested by COPYMULT, a large disk system may be downloaded to your present floppy disk system, any size. No need to fiddle with directory deletions or any of the other tedious operations that must be done using the normal copy routines.

COPYMULT.CMD understands normal "copy" syntax and always keeps up with files already copied by maintaining directories for both host and receiving disk system, eliminating hours of tedious keyboard entries and other time consuming cleanup chores.

BACKUP.CMD is a special program that downloads "random" type files, any size.

RESTORE.CMD a special program to restructure copied "random" files for copying, or recopying back to the host system.

FREELINK.CMD a "bonus" utility that "relinks" the free chain of floppy or hard disk thereby eliminating fragmentation.

Completely documented source files included.

ALL 4 Programs (FLEX, 8" or 5") $99.50

Southeast Media

XDATA

A COMMUNICATION Package
for the UniFLEX Operating System

Allows UniFLEX Based Systems to Transmit and Receive files to and from other Computer Systems via Modem. Use with CP/M, Main Frames, other UniFLEX Systems, etc.

-- Verifies Transmission integrity using checksum or CRC
-- Automatically Re-Transmits bad blocks
-- Transmits data in 128 byte blocks

U - $299.99

Southeast Media

OS-9 VDisk

Give your OS-9 Level I System the **speed of memory** access that can be several orders of magnitude over your present floppy disk drive. Use that **Extended Memory** capability of your SWTPC or Gimix CPU card (or any other that has the same format DAT). The size of the Virtual Disk is completely variable in whole increments of 4K up to 960K, which is all that these systems can address beyond the base page that OS-9 Level I uses. By putting all of your CMDS Directory on your Virtual Disk, you can have the fastest execution speed possible (next to eating up System Memory with all of them). You can also set up high speed inter-process communications via random virtual disk files and not eat up valuable system memory with pipe buffers. Some Assembly Required - **Level I ONLY.**

O. obj. only - $79.95
w/ Source - $149.95

Lucidata

PASCAL UTILITIES
Requires LUCIDATA Pascal ver 3.

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

F and CCF - $25.00

INCLUDE -- allows the inclusion of other Files in a Source Text; has unlimited nesting capabilities. Also allows Binary File inclusions.

F and CCF - $25.00

PROFILER -- produces an Indented, Numbered, "Structogram" of a Pascal Source Text File. Allows viewing the overall structure of large programs, and provides clues as to the integrity of the program. Supplied as Source Code; requires compilation.

F and CCF - $25.00

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

☎ TOLL FREE
1-800-338-6800
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
**OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

Lucidata

COPYCAT
Pascal NOT required

Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform Miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Includes Utilities to List Directories, Copy Files, and convert Text Files when required. Also includes a Utility for investigating Physical Compatibility problems. Programs supplied in **Modular Source Code** (Assembly Language) to make it easier to solve unusual problems.

Computer Systems Consultants

FLEX DISK UTILITIES

Eight (8) different FLEX Utilities that should be a part of every FLEX Users Toolbox; Assembly Language (Source Code):

Copy a File with CRC **Errors**, so it can possibly be salvaged; **Test Disk** for errors; **Compare two Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort and Replace** the Disk Directory (in sorted order).

PLUS

Ten BASIC Programs to:

A **BASIC Resequencer** with EXTRAs over "RENUM"; works with ALL versions of FLEX BASIC AND the Precompiler, checks for missing label definitions, processes Disk to Disk instead of In Memory.

Compare, Merge, or Generate Updates between two BASIC Programs, check **BASIC Sequence Numbers**, compare two unsequenced files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files.

A BASIC Cross-**Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC, XBASIC,** and **PRECOMPILER** BASIC Programs.

ALL Utilities **Include Source** (either BASIC or Source Code). An EXCELLENT Value!

F and CCF - $50.00

BUSINESS
WORD PROCESSING

Windrush Micro Systems

SCREDITOR III

EXTREMELY Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; EXCELLENT Documentation (over 300 pages), including a full Tutorial Section to help you learn how to use the system. Features Cursor-based editing, dynamic Screen Formatting (what you see is what you get), Multi-Column display and editing, "decimal align" columns (AND add them up automatically, if wanted), define multiple keystroke macros, even and odd page number headers and footers, imbed printer control codes in text, full justification series of commands, full "help" support, store common command series on disk for future use, etc. Easy "Set-Up" (for example, you just hit the key you want to use for a specific function, such as "cursor up", and the System reads an stores that key - no digging into tech manuals for codes, etc.); use supplied "set-ups", or remap the keyboard to what you are used too. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

Southeast Media

SPELLB "Computer Dictionary"
OVER 120,000 words!

No more "Let your fingers do the walking through the Dictionary" while you are entering Text with your favorite Editor or Word Processor. SPELLB is more than just "another Spelling Checker"; it allows you to **look up a word** from within your Editor or Word Processor so that you **KNOW** it is right WHEN YOU TYPE IT IN with the **SPH.CMD** Utility (which operates in the FLEX Utility Space). Yes, it ALSO allows you to check and update the Text after you are finished; along with allowing you to ADD WORDS to the Dictionary, "Flag" questionable words in the Text for evaluation later, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of **Small Disk Storage** systems.

F and CCF - $129.95

Great Plains Computer Co.

STYLOGRAPH

A full-screen oriented WORD PROCESSOR -- (now runs on the **Data-Comp** and FHL **Color FLEX** Systems; uses the 51 x 24 Display Screens). Full screen display and editing (i.e., what you see is what you get); supports the Daisy Wheel proportional printers.

SPECIAL CCF - $195.00

F and O - $295.00 U - $395.00

SPELL

Fast Computer Dictionary.

F, CCF, OS/9 - $125.00 U - $175.00

MAIL MERGE

Greatly extends the power and flexibility of STYLOGRAPH.

F, CCF, O - $145.00 U - $195.00

Southeast Media

JUST
Text Formatter

JUST, a Text Formatter developed by Ron Anderson, provides numerous features which make it a valuable addition to any FLEX Users Software Library. JUST is designed for formatting Text Output for Dot Matrix Printers and provides many unique features:

-Output the "Formatted" Text to the Display for format analysis and change.

-Output the "Formatted" Text to a Text File for use with the supplied FPRINT.CMD for producing multiple copies of the Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (this Utility is very useful at other times also, and worth the price of the program by itself).

-**"User Configurable"** for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); provides for up to ten (10) imbedded "Printer Control Commands", such as Italics on and off, boldface on and off, etc.

-Automatic compensation for a "Double Width" printed line.

-Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc.

-Use with **ANY** Editor.

-Supplied with "Structured Source" (Windrush PL/9); easy to see the flow of the program.

F and CCF - $49.95

Availability Legends —

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

FREE DISKETTE WITH EVERY $50 PURCHASE

☎ TOLL FREE
1-800-338-6800
For Ordering

TELEX 558 414 PVT BTH

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

DATA BASE MANAGEMENT SYSTEMS

Westchester Applied Business Systems

XDMS

Possibly one of the most powerful Database Management Systems' available, this machine language program is small enough to operate on a single sided 5" disk, yet provides the speed of H.L. and power limited only by the user's imagination. This DMS supports Relational, Sequential, Hierarchical, and Random Access File Structures, and has Virtual Memory capabilities for those Giant Data Bases. XDMS Level I provides a functional "entry level" System which provides for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility which uses an English Language Command Structure in manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds several special "Utilities" which provide additional ease of working with the various structures, changing System Parameters, etc.

XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95
XDMS System Manual only - $24.95

ACCOUNTING PACKAGES

Great Plains Computer Co. and Universal Data Research, Inc. both have Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX ----

• • • • • • • • • • • • •
- - - - Call 800-338-6800 for more information - - - -
• • • • • • • • • • • • •

Computer Systems Consultants

FULL SCREEN INVENTORY/MRP

The Full Screen Inventory System provides a means of maintaining small inventories. Using a linked, keyed random file structure based upon the item field, it keeps the file in alphabetical order for easier inquiry. With the FIND command, the user may locate and/or print all records matching on partial or complete item, description, vendor, or attributes. Items in backorder or below minimum stock levels may be located and/or printed thru the same process. Printed output may be produced in item or vendor order. A materials requirement planning (MRP) capability for manufacturing environments is included to allow the maintenance and analysis of Hierarchical assembiles of items in the inventory file. It requires TSC's Extended BASIC.

F and CCF - $100.00, U - $150.00

The Virginia Company

BIZPACK

BIZPACK is used for storing accounting, numeric, and financial data which can then be used for planning, budgeting, forecasting, analyzing, etc. While "Electronic Spreadsheets" are extremely useful in many situations, BIZPACK excels in businesses where there are numerous expense columns, revenue sources, significant business indicators, large numbers, erratic week-to-week and month-to-month fluctuations, etc. BIZPACK helps determine statistical relationships, establish trend lines, "smooths" data via moving averages, analyze seasonal data, adjusts for inflation, lags data in Statistics or Column functions, plots data, etc. BIZPACK is oriented toward time series analysis of businesses. The Program displays information on the screen in Columns of information with each Row conforming to a defined Period of Time (weeks, months, years, etc.), and is very easy to use (data is easy to enter, change, and modify; commands can be renamed to suit the users requirements; unlimited ability to create specialized commands using common BASIC Statements; etc.). Requires TSC's Extended BASIC.

F and CCF - $135.00
with Source - $250.00

Computer Systems Consultants

TABULA RASA SPREADSHEET

TABULA RASA is similar to DESKTOP/PLAN and provides for the generation and maintenance of tabular computation schemes often used for analysis of business, sales, and economic scenarios. Its menu-driven user interface provides these capabilities even to those users with no programming experience. Its extensive report-generation capabilities allow the user to generate professional results with minimum effort. It requires TSC's Extended BASIC.

F and CCF - $100.00, U - $200.00

Computer Systems Center

DYNACALC

THE Electronic Spread Sheet for 6809 Computer Systems. An extremely POWERFUL Business Tool, this Program will find an unlimited number of "non-business" applications, also (for example, a full Junior College Electronics Curriculum was set up using DYNACALC). Advanced features like "Table Lookup" make Income Tax work easy; Column or Row Sorting for numerous applications; etc. Completely "Memory Resident", Machine Language, this Program is FAST. Provides STANDARD FLEX Text File output for use with BASIC, Word Processors, Pascal, "C", etc. Also available for Data-Comp and FHL FLEX systems using the 50 x 24 Displays.

F and SPECIAL CCF - $200.00
U - $395.00

ODDS AND ENDS

Computer Systems Consultants

FULL SCREEN FORMS DISPLAY

This package supports any Serial Terminal with cursor control of Memory-Mapped Video Displays. The package substantially extends the screen Input/Output capabilities of TSC's Extended BASIC programs by providing a simple, table-driven method of describing and using full screen displays. These table entries are easy to set up and maintain, and are normally stored on disk and read as required. A simple, interactive means of generating the forms and the data field definitions is provided.

F and CCF - $50.00, U - $75.00

Computer Systems Consultants

FULL SCREEN MAILING LIST

The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Using a random fill structure based on the first character of the name field, it maintains the file in alphabetical order for easier inquiry. With the FIND command, the user may locate all records matching on partial or complete name, city, state, zip, or attributes. Printed listings and output to labels may also be produced on the same selective basis. It requires TSC's Extended BASIC.

F and CCF - $100.00, U - $110.00

VISA

MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

☎ TOLL FREE
1-800-338-6800
For Ordering
SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

Southeast Media

CHESS 6809

Requires FLEX and DISPLAYS **On Any Type Terminal**
Features:

*Four levels of play.
*Swap side. *Point scoring system.
*Two display boards. *Change skill level.
*Solve Checkmate problems in 1-2-3-4 moves.
*Make move and swap sides. *Play white or black.

This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels).

F and CCF - $79.95

Southeast Media

DIET-TRAC Forecaster

DIET-TRAC Forecaster is an XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable. bread, meat, skim milk, fruit and fat) for a specific individual.

Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. When a weight goal is given (either gain or loss), and a calorie plan is agreed upon between the computer and the individual, the number of days to reach the weight goal is projected. The starting and ending rate of weight loss is calculated, and a daily calendar with each day's weight for a 30-day period is printed.

F - $59.95
U - $89.95

COLOR COMPUTER SOFTWARE

Stearns Electronics

FORTH

Intrigued by Forth??? Here is a FORTH package tailored to the Color Computer! This package is supplied on Tape, with instructions for transferring it to disk if you wish. Written primarily in machine language, it's speed is unparalleled. A full Semigraphic-8 Editor is provided, along with "goodies" like Graphics and Sound Commands, Printer Commands, Auto-Repeat and Control Keys, etc. If you are Interested in Learning FORTH, a Trace Feature is provided which is invaluable. If you are a FORTH Pro, this package provides CPU carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. (Or; you won't "out grow" the Basic capabilities of this Implementation). Combine this package with Leo Brodie's EXCELLENT Book "Starting FORTH", and you will be a FORTH Expert before you know it (and have a lot of fun doing it!).

Color Computer TAPE - $58.95

Custom Software Engineering, Inc.

Color Computer **GRAPHIC SCREEN PRINT** Programs

Dumps any "PMODE" Screen to the Printer with the BASIC USR Function. Shift the Printout Left or Right or Reverse Print (Dark for Light Screen and Vice Versa). All Programs on Tape.

GSPR	for R.S. LP-VII/VIII & DMP 100/200/400	$7.95
GSPRE	for Epson w/ Graftrax and Graftrax +	$9.95
GSPRG	for Gemini 10 and 15	$9.95
GSPRP	for the Prowriter Printers	$9.95

Custom Software Engineering, Inc.

DATE-O-BASE CALENDAR Program

A Menu Driven **EXTENDED BASIC** Program which allows the entry of up to 12 Memos per Day, each of which may contain up to 28 Characters, for any day of the Month between the years 1700 and 2099. A **Graphic Calendar** shows which days contain Memos, and a "Key Word" Search is provided which can be output to the Screen or Printer.

TAPE DATE-O-BASE CALENDAR
(Each Tape File will hold up to 400 Memos) $16.95
DISK DATE-O-BASE CALENDAR
(4,000 Memos at 300/Month per Disk) $19.95

FREE DISKETTE WITH EVERY $50 PURCHASE

☎ **TOLL FREE**
1-800-338-6800
For Ordering

TELEX 558 414 PVT BTN

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

Custom Software Engineering, Inc.

That's INTEREST-ing

Interested in INTEREST (the Money Kind)? An EXTENDED BASIC Program that will help you deal with numerous problems requiring interest calculations. Present Value, Rate of Return, Current Bond Yield and Rate of Return to maturity, Loan Repayment Amortization Schedules, etc.

TAPE - $29.95

Custom Software Engineering, Inc.

DISK DATA HANDLER 64K

An EXTENDED BASIC Data Management System w/ Mach. Lang. Routines. Allows a max of 246 Chars. and 14 Fields per Record, and another Record can be linked to the first; 8 Char. Field Names, up to 99 Chars. per Field. Powerful On-Screen editor for input and update. flexible Output capabilities including output to Disk Files for use by other Programs. Change File Definition without re-entering the Data, Split Files, etc. Allows Multiple Field Sorts, Select on any combination of Fields, etc. An extremely POWERFUL TOOL; instructions provide examples of Mailing Lists and a Financial Stock Profit and Loss Tracking System.

DISK - $54.95

Custom Software Engineering, Inc.

DISK DOUBLE ENTRY

DISK EXTENDED BASIC Accounting Program w/ Mach. Lang. Routines. A "Traditional" Accounting Package for Small Business, Clubs, Churches, Personal Use, etc. Up to four levels of subtotals with Trial Balance, Income Statement, and Balance Sheet Reports. DDE allows up to 300 accounts and a Trial Balance of $9,999,999.99. Transactions may be up to 14 lines long, and comments and explanations may be freely used. Accounts are traceable to the journal transaction, which may include comments. Screen reports allow review of past transactions and current balances.

DISK - $44.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

☎ TOLL FREE
1-800-338-6800
For Ordering
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends ---
P = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

TEN MOST-ASKED QUESTIONS about DYNACALC™

THE ELECTRONIC SPREAD-SHEET FOR 6809 COMPUTERS

1. What is an electronic spread-sheet, anyway?
Business people use spread-sheets to organize columns and rows of figures. DYNACALC simulates the operation of a spread-sheet without the mess of paper and pencil. Of course, corrections and changes are a snap. Changing any entered value causes the whole spread-sheet to be re-calculated based on the new constants. This means that you can play, 'what if?' to your heart's content.

2. Is DYNACALC just for accountants, then?
Not at all. DYNACALC can be used for just about any type of job. Not only numbers, but alphanumeric messages can be handled. Engineers and other technical users will love DYNACALC's sixteen-digit math and built-in scientific functions. You can build worksheets as large as 256 columns or 256 rows. There's even a built-in sort command, so you can use DYNACALC to manage small data bases — up to 256 records.

3. What will DYNACALC do for ME?
That's a good question. Basically the answer is that DYNACALC will let your computer do just about anything you can imagine. Ask your friends who have VisiCalc™, or a similar program, just how useful an electronic spread-sheet program can be for all types of household, business, engineering, and scientific applications. Typical uses include financial planning and budgeting, sales records, bills of material, depreciation schedules, student grade records, job costing, income tax preparation, checkbook balancing, parts inventories, and payroll. But there is no limit to what YOU can do with DYNACALC.

4. Do I have to learn computer programming?
NO! DYNACALC is designed to be used by non-programmers, but even a Ph.D. in Computer Science can understand it. Even experienced programmers can get jobs done many times faster with DYNACALC, compared to conventional programming. Built-in HELP messages are provided for quick reference to operating instructions.

5. Do I have to modify my system to use DYNACALC?
Nope. DYNACALC uses any standard 6809 configuration, so you don't have to spend money on another CPU board or waste time learning another operating system.

Order your DYNACALC today!

Foreign Dealers:

Australia & Southeast Asia: order from Paris Radio Electronics, 161 Bunnerong Road (PO Box 380) Kingsford, 2032 NSW Australia. Telephone: 02-344-9111.

United Kingdom: order from Compusense, Ltd., PO Box 169, London N13 4HT. Telephone: 01-882-0681.

Scandinavia: order from Swedish Electronics hk AB, Murargatan 23-25, Uppsala 5-754 37 Sweden. Telephone: 18-25-30-00.

6. Will DYNACALC read my existing data files?
You bet! DYNACALC has a beautifully simple method of reading and writing data files, so you can communicate both ways with other programs on your system, such as the Text Editor, Text Processor, Sort/Merge, STYLOGRAPH™ word processor, RMS™ data base system, or other programs written in BASIC, C, PASCAL, FORTRAN, and so on.

7. How fast is DYNACALC?
Very. Except for a few seldom-used commands, DYNACALC is memory-resident, so there is little disk I/O to slow things down. The whole data array (worksheet) is in memory, so access to any point is instantaneous. DYNACALC is 100% 6809 machine code for blistering speed.

8. Is there a version of DYNACALC for MY system?
Probably. You need a 6809 computer (32k minimum) with FLEX™, UniFLEX™, or OS-9™ operating system. You also need a decent crt terminal, one with at least 80 characters per line, and direct cursor addressing. If your terminal isn't smart enough for DYNACALC, you probably need a new one anyway. The UniFLEX and OS-9 versions of DYNACALC allow you to mix different brands of terminal on the same system. There's also a special version of DYNACALC for Color Computers equipped with FLEX (Frank Hogg or Data-Comp versions).

9. How much does DYNACALC cost?
The FLEX versions are just $200 per copy; UniFLEX version $395; OS-9 version (works with LEVEL ONE or LEVEL TWO) $250. Orders outside North America add $7 per copy for postage. We encourage dealers to handle DYNACALC, since it's a product that sells instantly upon demonstration. Call or write on your company letterhead for more information.

10. Where do I order DYNACALC?
See your local DYNACALC dealer, or order directly from CSC at the address below. We accept telephone orders from 10 am to 6 pm, Monday through Friday. Call us at 314-576-5020. Your VISA or MasterCard is welcome. Please specify diskette size for FLEX or OS-9 versions. Software serial number is required for the UniFLEX version.

Computer Systems Center
13461 Olive Blvd.
Chesterfield, MO 63017
(314) 576-5020

CSC

UniFLEX software prices include maintenance for the first year.

DYNACALC is a trademark of Computer Systems Center

VisiCalc is a trademark of VisiCorp.
STYLOGRAPH is a trademark of Great Plains Computer Co.
RMS is a trademark of Washington Computer Services.
FLEX and UniFLEX are trademarks of TSC.
OS-9 is a trademark of Microware and Motorola.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, MCM68764/6, I2764/2764A, T2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL .... a selected area of the buffer with a HEX char.
b. MOVE .... blocks of data.
c. DUMP .... the buffer in HEX and ASCII.
d. FIND .... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC .... checksum a selected area of the buffer.
g. COPY .... a selected area of an EPROM into the buffer.
h. VERIFY .... a selected area of an EPROM against the buffer.
i. PROGRAM .... a selected area of an EPROM with data in the buffer.
j. SELECT .... a new EPROM type (return to types menu).
k. ENTER .... the system monitor.
l RETURN .... to the operating system.
1. EXECUTE .... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, REXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

WORSTEAD LABORATORIES, NORTH WALSHAM,
NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:

- Talker	- Serial Poll	- Single or Dual Primary Address
- Listener	- Parallel Poll	- Secondary Address
- System Controller	- Group Trigger	- Talk only ... Listen only

- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface) ..	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	SS-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II ...	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

WE STOCK THE FOLLOWING COMPANIES PRODUCTS:
GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

FEATURES THE POWERFUL, THIRD GENERATION, MOTOROLA 6809 PROCESSOR!

THE 6809 "UNIBOARD"™

SINGLE BOARD COMPUTER KIT

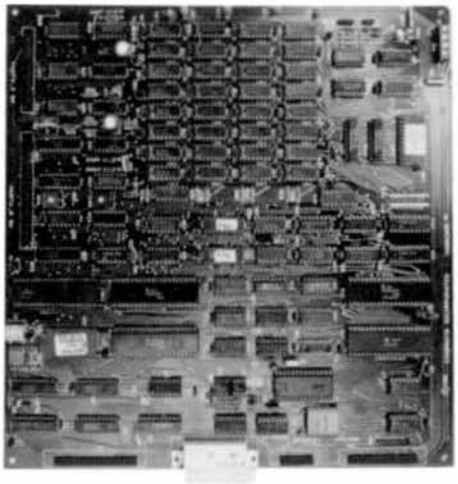
PERFECT FOR COLLEGES, OEM'S, INDUSTRIAL AND SCIENTIFIC USES!

64K RAM! DOUBLE DENSITY FLOPPY DISK CONTROLLER!

New! Lower Price!

BLANK PC BOARD
$99^95
WITH PAL'S, AND TWO EPROMS.

FOR 5-1/4 OR 8 INCH SOURCE DISKETTE ADD $10.



$249^00
COMPLETE KIT! FULLY SOCKETED.

PRICE CUT!!

THE COMPACTA UNIBOARD™: Through special arrangement with COMPACTA INC., we are proud to have been selected the exclusive U.S. Mfg. of their new 6809 UNIBOARD™ COMPUTER KIT. Many software professionals feel that the 6809 features probably the most powerful instruction set available today on ANY 8 bit micro. Now, at last, all of that immense computing power is available at a truly unbelievably low price.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A FREE COPY IS AVAILABLE UPON REQUEST.

FEATURES:

- ★ 64K RAM using 4116 RAMS.
- ★ 6809E Motorola CPU.
- ★ Double Density Floppy Disk Controller for either 5-1/4 or 8 inch drives. Uses WD1793.
- ★ On board 80 x 24 video for a low cost console. Uses 2716 Char. Gen. Programmable Formats. Uses 6845 CRT Controller.
- ★ ASCII keyboard parallel input interface. (6522)
- ★ Serial I/O (6551) for RS232C or 20 MA loop.
- ★ Centronics compatible parallel printer interface. (6522)
- ★ Buss expansion interface with DMA channel. (6844)
- ★ Dual timer for real time clock application.
- ★ Powerful on board system monitor (2732). Features commands such as Go To, Alter, Fill, Move, Display, or Test Memory. Also Read and Write Sectors. Boot Normal, Unknown, and General Flex™.

YOUR CHOICE OF POPULAR DISK OPERATING SYSTEMS:

FLEX™ from TSC	$99
OS9™ from Microware	$199

Specify 5-1/4 or 8 inch

PC BOARD IS DOUBLE SIDED, PLATED THRU SOLDER MASKED, 11 x 11-1/2 IN.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Shipments will be made approximately 3 to 6 weeks after we receive your order. VISA, MC, cash accepted. Add $4.00 shipping. USA AND CANADA ONLY

64K SS-50 STATIC RAM

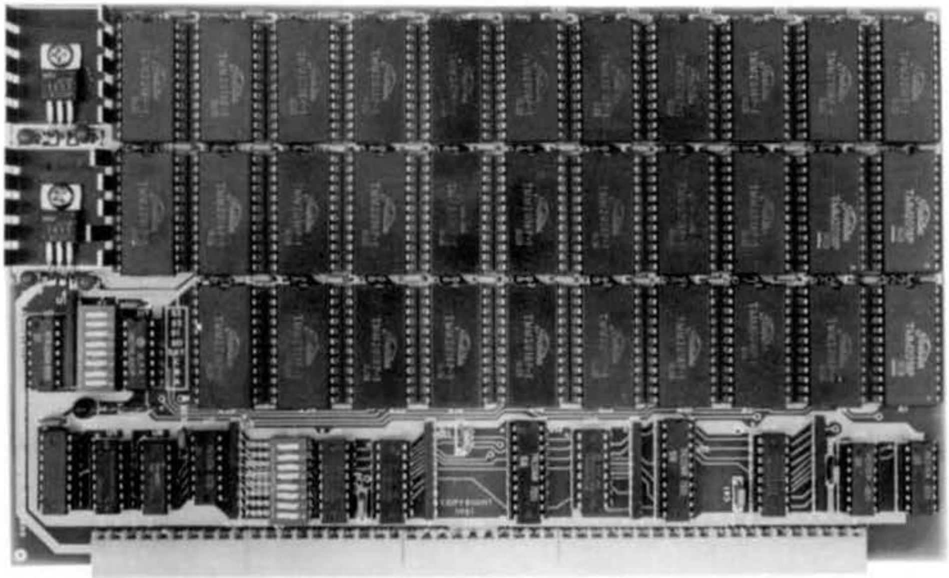
PRICE CUT!!

$149.00 (48K KIT)

NEW!

LOW POWER!

RAM OR EPROM!



BLANK PC BOARD
WITH DOCUMENTATION
$45

SUPPORT ICs + CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $50**

FEATURES:

★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
★ Fully supports Extended Addressing.
★ 64K draws only approximately 500 MA.
★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
★ 2716 EPROMs may be installed anywhere on Board.
★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
★ One Board supports both RAM and EPROM.
★ RAM supports 2MHZ operation at no extra charge!
★ Board may be partially populated in 16K increments.

56K	$169
64K	$199

CLOSE OUT SPECIAL
WE HAVE DROPPED OUR 32K SS-50 STATIC RAM BOARD WHICH USED 2114 LOW POWER RAMS. WE WILI. SELL THE REMAINING STOCK OF BLANK PCB'S WITH DATA FOR $17.50 EA. THESEFORMERLY SOLDFOR $50.

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

DISKETTES AND 680X SOFTWARE

SUPER SLEUTH DISASSEMBLER EACH $99-FLEX, $101-OS-9, $100-UNIFLEX
interactively generates source on disk with labels, includes xref, label definition, binary file editing, etc.
specify 6800,1,2,3,5,8,9/6502 version or Z-80/8080/85 version
OS-9 and UNIFLEX versions also process FLEX object file formats
OBJECT ONLY versions: EACH $50-FLEX & OS-9, $40-COCO DOS
COCO DOS available in 6800,1,2,3,5,8,9/6502 version only

CROSS-ASSEMBLERS EACH $50-FLEX/UNIFLEX/OS-9, ANY 3 $100, ALL $200
specify for 180x, 650x, 680x, Z-80, 8048/51, 8085, 68000
true, modular, free-standing cross-assemblers, written in C
8-bit source included only with all cross-assemblers (for $200)

DEBUGGING SIMULATORS EACH $75-FLEX, $100-OS-9, $80-UNIFLEX
specify 6800/1, (14)6805, 6502, 6809 OS-9, Z-80 FLEX
OBJECT ONLY versi ns: EACH $50-COCO FLEX & COCO OS-9

6502 TO 6809 ASSEMBLER TRANSLATOR $75-FLEX, $85-OS-9, $80-UNIFLEX
translates 6502 programs to 6809, noting inexact conversions

6800 TO 6809 & 6809 PIC TRANSLATORS $50-FLEX, $75-OS-9, $60-UNIFLEX
translates 6800 programs to 6809, 6809 programs to PIC

FULL-SCREEN FLEX AND UNIFLEX TSC XBASIC PROGRAMS FOR 6809
(with complete cursor control)

DISPLAY GENERATOR/DOCUMENTOR	$50 w/source, $25 without
MAILING LIST SYSTEM	$100 w/source, $50 without
INVENTORY WITH MRP	$100 w/source, $50 without
TABULA RASA SPREADSHEET	$100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY $50-FLEX & UNIFLEX
edit sectors, sort directory, maintain master catalog, do disk sorts, xref BASIC, ...

CMODEM PROGRAM $100-FLEX & OS-9 & UNIFLEX, OBJECT-ONLY EACH $50
provides menu-driven telecommunications facilities, with terminal mode, up/down load, MODEM7 protocol, etc.

5.25" SOFT-SECTORED DISKS EACH 10 $13-SSSD $15-SSDD $17-DSDD $25-DSQD
with jackets and hub rings

SS-50C 256K 1.5MHZ MEMORY BOARDS BLANK $100 A&T $350
with instruction manual, schematics, and delay line; all parts readily available

Most programs in source on disk: specify computer, disk size, operating system,
Contact CSC for full catalog and dealer information.
25% discount for multiple purchases of same program on same order.
For VISA and MASTER CARD, give account, exp. date, phone. US funds only.
Add GA sales tax and 5% shipping; no shipping for disks in 100's.
(UNIFLEX trademark Technical Systems Consultants. OS-9 trademark Microware.

Computer Systems Consultants, Inc.
1454 Latta Lane, Conyers, GA 30207
Telephone Number 404-483-1717/4570

SOFTWARE FOR THE HARDCORE

** FORTH PROGRAMMING TOOLS from the 68XX&X **
** FORTH specialists — get the best!! **

NOW AVAILABLE — A variety of rom and disk FORTH systems to run on and/or do TARGET COMPILATION for

6800, 6301/6801, 6809, 68000, 8080, Z80

Write or call for information on a special system to fit your requirement.

Standard systems available for these hardware—

EPSON HX-20 rom system and target compiler
6809 rom systems for SS-50, EXORCISER, STD, ETC.
COLOR COMPUTER
6800/6809 FLEX or EXORCISER disk systems.
68000 rom based systems
68000 CP/M-68K disk systems, MODEL II/12/16

tFORTH is a refined version of FORTH Interest Group standard FORTH, faster than FIG-FORTH. FORTH is both a compiler and an interpreter. It executes orders of magnitudes faster than interpretive BASIC. MORE IMPORTANT, CODE DEVELOPMENT AND TESTING is much, much fa ter than complied languages such as PASCAL and C. If Software DEVELOPMENT COSTS are an important concern for you, you need FORTH!

firmFORTH™ is for the programmer who needs to squeeze the most into roms. It is a professional programmer's tool for compact rommable code for controller applications.

™ tFORTH and firmFORTH are trademarks of Talbot Microsystems
™ FLEX is a trademark of Technical Systems Consultants, Inc
™ CP/M-68K is trademark of Digital Research, Inc

tFORTH™
from TALBOT MICROSYSTEMS
NEW SYSTEMS FOR
6301/6801, 6809, and 68000

---> tFORTH SYSTEMS <---

For all FLEX systems: GIMIX, SWTP, SSB, or EXORcisor Specify 5 or 8 inch diskette, hardware type, and 6800 or 6809.

** tFORTH — extended fig FORTH (1 disk) with fig line editor. $100 ($15)
** tFORTH + — more! (3 5" or 2 8" disks) $250 ($25)
adds screen editor, assembler, extended data type , utilities, games, and debugging aids.
** TRS-80 COLORFORTH — available from The Micro Works
** firm FORTH — 6809 only. $350 ($10)
For target compilations to rommable code.
Automatically deletes unused code. Includes HOST system sour and target nucleus source. No royalty on target . Requires but does not include tFORTH + .
** FORTH PROGRAMMING AIDS — elaborate decompiler$150

** tFORTH for HX-20, in 16K roms for expansion unit or replace BASIC $170

** tFORTH/68K for CP/M-68K 8" disk system $290
Makes Model 16 a super software development system.

** Nautilus System Cross Compiler
— Requires: tFORTH + HOST + at least one TARGET:
— HOST system code (6809 or 68000) $200
— TARGET source code: 6800-$200, 6301/6801—$200
same plus HX-20 extensions— $300
6809—$300, 8080/Z80—$200, 68000—$350

Manuals available separately — price in ( ).
Add $6/system for shipping, $15 for foreign air.

TALBOT MICROSYSTEMS 1927 Curtis Ave., Redondo Beach, CA 90278 (213) 376 9941

!!! FREE !!!

Published Monthly by Computer Publishing Inc., Hixson, TN.

$1.95

Bulk Rate
U.S. Postage
PAID
Chattanooga, TN
Permit No. 357

Color Micro Journal

The Color Computer Monthly Magazine

$1.95 per issue Vol. 1, Issue 2 October, 1983

THIS 'N THAT

The BIG NEWS this month is that OS-9 has finally arrived for the Color Computer. The ASTOUNDING part of the Radio Shack OS-9 Package, besides the price, is the DOCUMENTATION. You 'Old Time Radio Shack Followers' will not believe what you see. Jon Shirley has been telling us that the main reason for the "lack" of documentation with a lot of their products was the restrictions placed on releasing that information by Microsoft; I

OS-9 on the COLOR COMPUTER

One of the 'Operating Systems of the Future" is now available for the "little old Color Computer"; OS-9. Freely translated, OS-9 means "Operating System for the 6809" (OS-9 is now being written for the 68000, also). Since it is fairly obvious that UNIX and "UNIX-Type" Operating Systems will be running on just about every computer to come out in the next few years, a whole new language is beginning to appear on the horizon.

Color Computer OS-9, the Package

We had been running a preliminary release of OS-9 on the Color Computer for a few weeks, and received the "Official Radio Shack" version for Review a couple of days ago. To put it mildly, this package is IMPRESSIVE! For $69.95 (Radio Shack Catalog Number 26-3030), you receive a 9 1/2" x 7 5/8" x 2" package containing 4

FREE SAMPLE ISSUE

1-800-338 6800

MON.-FRI. 9-5 E.S.T.

TELEX 558 414 PVT BTH

USA-$12.50 per year. Canada& Mexico-$19.50 per year

Surface Foreign-$24.50 per year. Airmail Foreign-$48.50 per year

Color Micro Journal™

TM Color Micro Journal is a trademark of Computer Publishing Inc.

5900 Cassandra Smith Rd. Hixson,TN. 37343

INTELLICOM™

An INTELLIigent COMmunications Program

Get in Touch with

- Easy Installation
- Menu Driven
- Intelligent computer to computer communications
- Supports most file transfer protocols
- Transfers CPM files to your system (Christensen Protocol)
- Access to timesharing services (Source, Compuserve)
- Available for OS/9 and Flex

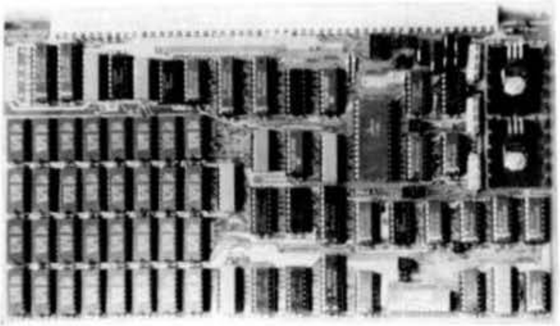
The World of Intellicom

Price: $ 99.95

Great Plains Computer Company
P. O. BOX 916
Idaho Falls, Idaho 83403
(208) 529-3210

OS9 is a trademark of Microware

Flex is a trademark of TSC, Inc



256K, 512K, 1 MEG MEMORY SYSTEM

Now compatible with DMA controllers. Runs at up to 2Mhz without generating MRDY or Interrupts. Has an optional on board DAT for use with CPU cards without a DAT. 128K, 256K, 512K or 1M byte per card. Field upgradable. Optional configuration allows 4M byte address reach (using memory board DAT) without CPU changes or cables. 1 year limited warranty.

TURBO virtual disk softwar and memory diagnostics supplied with the system.

Prepaid: 256K: $750.00, 128K: $595.00, 512K: $1095.00, 1024K: $1595.00

Domestic shipping and handling $10.00. Users manual: $15.00, applicable toward system purchase. Cashiers check, COD, personal checks must clear before shipment. Fla. residents add 5% sales tax. Shipped stock to 30 days. Dealer and quantity discounts available.

COMPUTER EXCELLENCE INC.
P.O. BOX 8442
CORAL SPRINGS, FL 33065
(305) 752-8321

COMPARE

our EPROM PROGRAMMER with the field.

All data taken directly from manufacturer's current advertising. Software, interfaces, or personality modules may also be required at additional cost.

- • Triple voltage EPROM
- • Supplied in kit form

		A	B	C	D	E	F
INTERFACE	S30	PAR	PAR	SER	S30	SER	SER
INTELLIGENT	NO	NO	NG	YES	NO	YES	YES
PROGRAMS							
2704•			●				●
2508	●			●	●	●	
2708•			●		●		●
2758	●	●	●	●		●	●
2516	●	●	●	●	●	●	
2716	●	●	●	●	●	●	●
2716•			●		●		●
2532	●		●	●	●	●	●
2732	●		●	●	●	●	●
2732A	●		●		●	●	●
2564	●		●		●	●	●
2764	●		●		●	●	●
2528	●				●		
27128	●						
2816							●
68764			●				
8748						●	
8749						●	
TOTAL	11	3	12	6	11	11	11
PRICE	$125	$45•	$169	$289	$375	$489	$575

EPRAM EPROM Programmer, $125. Personality module for 2508, 2758, 516, and 2716 included. Specify CPU, disk size, and operating system (TSC's FLEX or SSB's DOS) when ordering. Manual only, $10; refundable with EPRAM purchase.

UNITEK • P.O. Box 671 • Emporia, VA 23847

'68' MICRO JOURNAL

★ The only ALL 6800 Computer Magazine.

★ More 6800 material than all the others combined: MAGAZINE COMPARISON

(2 years)

Monthly Averages

	6800 Articles			TOTAL
KB	BYTE	CC	DOBB'S	PAGES
7.8	6.4	2.7	2.2	19.1 ea. mo.

Average cost for all four each month: $6.53
(Based on advertised 1-year subscription price)

'68' cost per month: $2.04

That's Right! Much, Much More

for About

1/3 the Cost!

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge □ — VISA □

Card #__________ Exp. Date__________

For □ 1-Year □ 2 Years □ 3 Years

Enclosed: $__________

Name____________________

Street____________________

City__________ State________ Zip______

My Computer Is:____________________

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

SUBSCRIPTION RATES

USA
1 Year $24.50, 2 Year $42.50, 3 Year $64.50

*FOREIGN SURFACE Add $12.00 per Year to USA Price

*FOREIGN AIRMAIL Add $36.00 per Year to USA Price

**CANADA & MEXICO Add $5.50 per Year to USA Price
Cash (USA) or drawn on a USA Bank!!!

DYNAMITE+™

"THE CODE BUSTER"

disassembles any 6809 or 6800 machine code program into beautiful source

- Learn to program like the experts!
- Adapt existing programs to your needs!
- Convert your 6800 programs to 6809!
- Automatic LABEL generation.
- Allows specifying FCB's, FCC's, FDB's, etc.
- Constants input from DISK or CONSOLE.
- Automatically uses system variable NAMES.
- Output to console, printer, or disk file.
- Available for all popular 6809 operating systems.

FLEX™ $100 per copy; specify 5" or 8" diskette.
OS-9™ $150 per copy; specify 5" or 8" diskette.
UniFLEX™ $300 per copy; 8" diskette only.

For a free sample disassembly that'll convince you DYNAMITE+ is the world's best disassembler, send us your name, address, and the name of your operating system.

Order your DYNAMITE+ today!

See your local DYNAMITE+ dealer, or order directly from CSC at the address below. We accept telephone orders from 10 am to 6 pm, Monday through Friday. Call us at 314-576-5020. Your VISA or MasterCard is welcome. Orders outside North America add $5 per copy. Please specify diskette size for FLEX or OS-9 versions.

Foreign Dealers:

Australia & Southeast Asia: order from Paris Radio Electronics, 161 Bunnerong Road (PO Box 380) Kingsford, 2032 NSW Australia. Telephone: 02-344-9111.

United Kingdom: order from Compusense, Ltd., PO Box 169, London N13 4HT. Telephone: 01-882-0681.

Scandinavia: order from Swedish Electronics hk AB, Murargatan 23-25, Uppsala S-754 37 Sweden. Telephone: 18-25-30-00.

Computer Systems Center
13461 Olive Blvd.
Chesterfield, MO 63017
(314) 576-5020

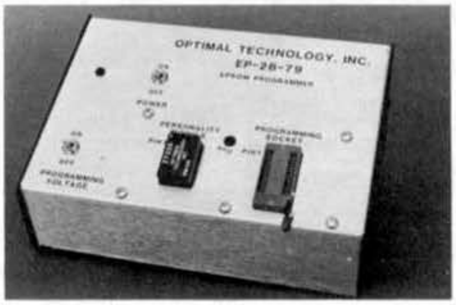
UniFLEX software prices include maintenance for the first year.

DYNAMITE+ is a trademark of Computer Systems Center.

FLEX and UniFLEX are trademarks of TSC.
OS-9 is a trademark of Microware and Motorola.

Dealer inquiries welcome.

Model EP-2B-79
EPROM Programmer



PROGRAMS 2764A, 27128A, 27256 in typically 45, 90, and 180 seconds 2764, 27128 in typically 50, and 100 seconds.

SUPPORTS 2816A, 2864A EEPROMs

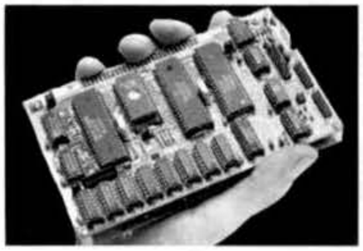
Other devices supported!
2708, 2716, 27C16, 2732, 27C32, 2732A, 2758, MCM68766, 2764A, 2764, 27C64, 27128, 27128A, 27256, 27C256, 2816A, 2864A, R87C32, 8751, 38E70, 8748H, and 8749H.

New software, EPROC 9.019 operating under FLEX™ allows the user to load from disk, offset load, save to disk, Program, verify, read to memory, execute FLEX and MONITOR commands, Operate from any I/O slot and many more easy to use features. Truly an elegant solution for both the experienced and novice programmers.

EP-2B-79 $179.00, Software $30, I/O Interface $39. Hardware upgrade for EP-2A-79 $25. Personality modules priced $17 to $35.

Optimal Technology, Inc.

Phone (804) 973-5482
Blue Wood 127 Earlysville, VA 22936



NEW!

Compact Flexible 6809 Computer

The new ST-2900 system — a complete 64K small business or hobbyist computer is only one of its many possible configurations. Among its features are:

- Small enough to hold in your hand! (Eurocard size: 3.9" x 6.3")
- Two board "system" for greater versatility than single board computers.
- CPU Board — powerful 6809E processor, 16K or 64K RAM, 2K-8K EPROM, 2 RS232 serial ports with software pr grammable baud rates, 16 bit counter/timer. Run the CPU board all by itself, or plug your own custom board or our FDC board into the expansion connector.
- FDC Board — double-sided/double-density floppy disk controller with adjustment free digital data separator and write precompensation, 2-8 bit parallel ports, 2-16 bit counter/timers, prototyping area.
- Available as bare PC boards or partially assembled boards. All have solde mask both sides plus silkscreened comp nent overlay.
- **OS-9 for only $49?**
  Well, not quite. But that's all you pay for our OS-9 Conversion Package which lets y u use the low cost Radio Shack CoCo version of OS-9 on our ST-2900 system. Save $131 off the suggested list price of OS-9! Supports CoCo OS-9 and standard OS-9 format disks.

• CPU bare ard plus EPROM	$45	FLEX Conversion Package	$29
FDC bare board	$38	CPU + FDC + OS-9 C nversion	$119

- Add $5 shipping/handling ($10 overseas). These prices are in U.S. funds. Canadian orders: call or write for prices. Terms: money order, certified check, VISA.

(FLEX is a trademark of Technical Systems Consultants. OS-9 is a trademark of Microware and Motorola)

Write for free brochure and complete price list.
(604) 255-4485
(4 - 5 pm Pacific Time)

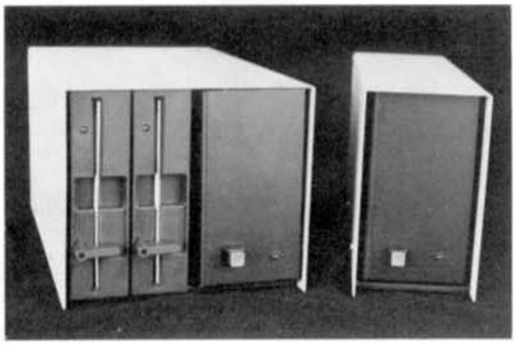
SARDIS TECHNOLOGIES

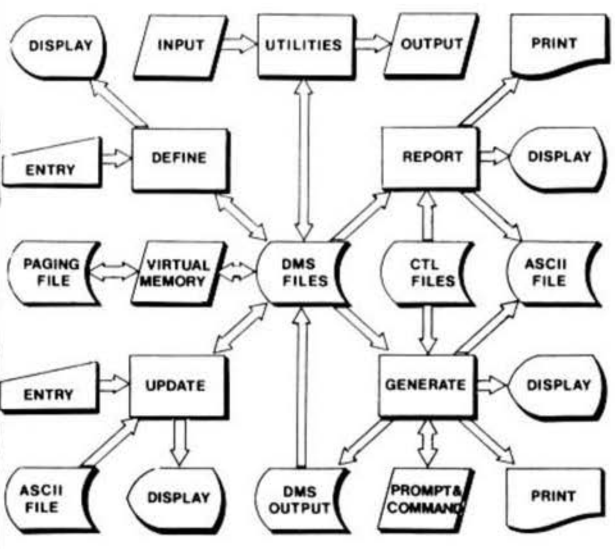
2261 E. 11th Ave. Vancouver, B.C., Canada V5N 1Z7

68' MICRO JOURNAL ADVERTISERS INDEX

'68' MICRO JOURNAL 51,68
ACORN COMPUTER SYSTEMS 70
ANDERSON COMPUTER CONSULTANTS 68
COLOR MICRO JOURNAL 65
COMPILER EVALUATION SERVICES 52
COMPUTER EXCELLENCE INC. 66
COMPUTER PUBLISHING INC. 5
COMPUTER SYSTEMS CENTER 60,69
COMPUTER SYSTEMS CONSULTANTS, INC. ...64
DATA-COMP ....IBC
DIGITAL RESEARCH COMPUTERS 62,63
GIMIX, INC. 3,72
GREAT PLAINS COMPUTER CO. 66
HAZELWOOD COMPUTER SYSTEMS ....OBC
INTROL CORP. 53
LLOYD I/O 52
MEASUREMENT TECHNOLOGIES 51
MICROKEY LTD. 51
MICROWARE SYSTEMS CORP. 1,4
OPTIMAL TECHNOLOGY INC. 69
PERIPHERAL TECHNOLOGY 52,71
SARDIS TECHNOLOGIES 69
SMOKE SIGNAL BROADCASTING 6
SOUTH EAST MEDIA 54,55,56,57,58,59
SOUTHWEST TECHNICAL PRODUCTS INC. ...IFC
STAR-KITS 68
TALBOT MICROSYSTEMS 64
TMP SOFTWARE 67
UNITEK 66
WESTCHESTER APPLIED BUSINESS SYSTEMS .71
WINDRUSH MICRO SYSTEMS LIMITED 61

This index is provided as a reader service. The publisher does not assume any liability for omissions or errors.




DISPLAY
INPUT
UTILITIES
OUTPUT
PRINT
ENTRY
DEFINE
REPORT
DISPLAY
PAGING FILE
VIRTUAL MEMORY
DMS FILES
CTL FILES
ASCII FILE
ENTRY
UPDATE
GENERATE
DISPLAY
ASCII FILE
DISPLAY
DMS OUTPUT
PROMPT& COMMAND
PRINT

GIMIX HAS THE 6809 SYSTEM TO SUIT YOUR NEEDS

HARDWARE

All systems feature the **GIMIX CLASSY CHASSIS**; with a ferro-resonant constant voltage power supply, gold plated bus connectors, and plenty of capacity for future expansion.

Static **RAM** and double-density **DMA** floppy disk controllers are used exclusively in all systems.

All systems are guaranteed for 2 MHz operation and include complete hardware and software documentation, necessary cables, filler plates, etc.

Systems are assembled using burned-in and tested boards, and all disk drives are tested and aligned by **GIMIX**.

You can add additional components to any system when ordering, or expand it in the future by adding **RAM, I/O**, etc.

GIMIX lets you choose from a wide variety of options to customize your system to your needs.

SOFTWARE

All **OS-9/FLEX** systems allow you to software select either operating system.

Also included is the **GMXBUG** monitor and, in systems with 128K or more of RAM, **GMX-VDISK** for **FLEX**.

All **GIMIX OS-9** systems include **Microware's Editor, Assembler, Debugger, Basic09,** and **Runb**; and the GMX versions of **RMS** and **DO** for **OS-9**.

All **GIMIX** versions of **OS-9** can read and write RS color computer format OS-9 disks, as well as the Microware/GIMIX standard format.

New and exclusive with **OS-9 GMX III** systems is the **GMX OS-9 Support ROM**, a monitor for **OS-9** that includes memory diagnostics and allows the system to boot directly from either hard disk or floppy.

A wide variety of languages and other software is available for use with either **OS-9** or **FLEX**.

OS-9 GMX III/FLEX SYSTEMS (#79)

The #79 super system now Includes (in addition to the above): the **GMX 6809 CPU III**, a **256K CMOS Static RAM Board (#72)**, and a **3-port Intelligent Serial I/O Processor (#11)**.

The **GMX 6809 CPU III** can perform high-speed DMA transfers from memory to memory and uses memory attributes and illegal instruction trapping to protect the system and users from program crashes. If a user program crashes, only that user is affected; other users are unaware of the problem.

The 3-Port Intelligent Serial I/O Board (#11) significantly reduces system overhead by handling routine I/O functions, freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud.

with dual 40 track DSDD drives	$5998.79
with dual 80 track DSDD drives	$6198.79
with **#88** dual 8" DSDD drive system	$7698.79
with **#90** 19MB Winchester subsystem and one 80 track	$8898.79
with a 47MB Winchester subsystem and one 80 track	$10,898.79
with a 47MB plus a 6 MB removable pack Winchester subsystem and one 80 track drive	$12,398.79

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account #73-32033.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc. FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc. GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

OS-9 GMX I / FLEX SYSTEMS #49

The #49 systems include 64KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track DSDD drives	$3998.49
with dual 80 track DSDD drives	$4198.49
with **#88** dual 8" DSDD drive system	$5698.49
with **#90** 19MB Winchester subsystem and one 80 track	$6898.49

OS-9 GMX II / FLEX SYSTEMS #39

The #39 systems include 128KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track DSDD drives	$4498.39
with dual 80 track DSDD drives	$4698.39
with **#88** dual 8" DSDD drive system	$6198.39
with #90 19MB Winchester subsystem and one 80 track	$7398.39

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

ALL PRICES ARE F.O.B. CHICAGO

Contact GIMIX for price and availability of UniFLEX and UniFLEX GMXIII Systems.

NOTE on all drive systems: Dual 40 track drives have about 700KB of formatted capacity; dual 80's about 1,400KB; dual 8" about 2,000KB. The formatted capacity of hard disks is about 80% of the total capacity.

Want to expand your system to a megabyte of Static RAM and 15 users?

Simply add additional memory and I/O boards. Your GIMIX system can grow with your needs. Contact us for a complete list of available boards and options.

#72 256KB CMOS STATIC RAM board with battery back up	$1898.72
#64 64KB CMDS STATIC RAM board with battery back up	$528.64
#67 64KB STATIC RAM board	$478.67
#11 3 port intelligent serial I/O board	$498.11
#43 2 port serial I/O board	$128.43
#42 2 port parallel I/O board	$88.42
#95 cable sets (1 needed per port). specify board	$24.95

NOW SHIPPING !

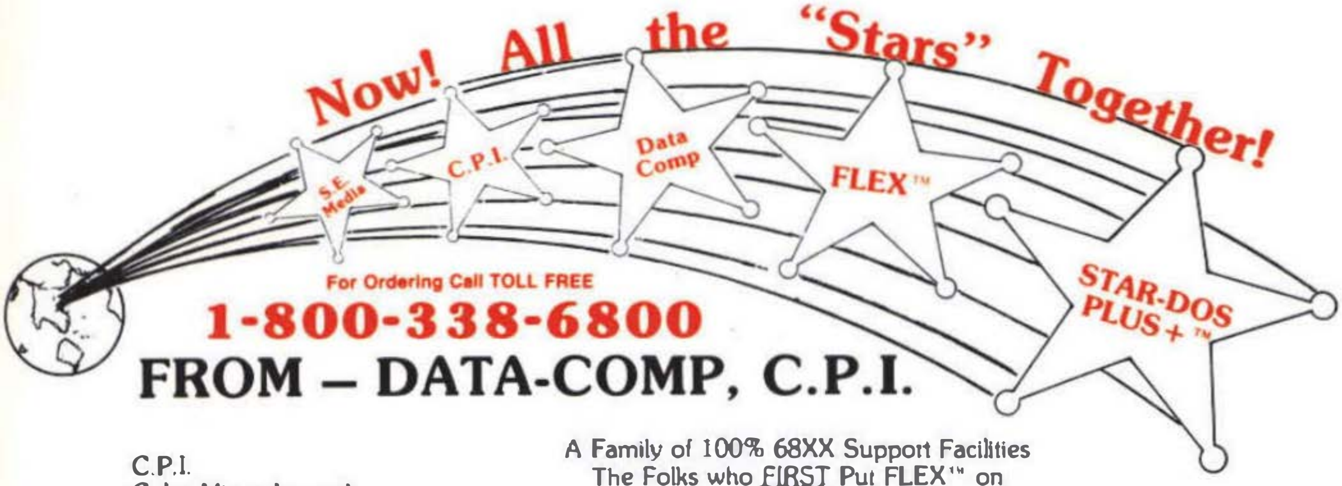
UniFLEX

GMX III Systems

GIMIX inc.

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

Now! All the "Stars" Together!



For Ordering Call TOLL FREE

1-800-338-6800

FROM – DATA-COMP, C.P.I.

C.P.I.
Color Micro Journal
'68' Micro Journal
Data-Comp
S.E. Media

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo
Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00
Only $79.95

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

TSC Editor
Reg $50.00
NOW $35.00

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ Super 800 Support
+ Free Color Micro Journal 1 yr. sub.
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Assembler
Reg $50.00
NOW $35.00

DISK SYSTEMS FOR THE COLOR COMPUTER

THESE PACKAGES INCLUDE DRIVE, *CONTROLLER, POWER SUPPLY & CABINET, CABLE, AND MANUAL.

* SPECIFY WHAT CONTROLLER YOU WANT J&M, OR RADIO SHACK.

PAK #1 - 1 SINGLE SIDED, DOUBLE DENSITY SYS.	$389.95
PAK #2 - 2 SINGLE SIDED, DOUBLE DENSITY SYS.	$639.95
PAK #3 - 1 DOUBLE SIDED, DOUBLE DENSITY STS.	$439.95
PAK #4 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS.	$699.95
PAK #5 - 2 DOUBLE SIDED, DOUBLE DENSITY STS. THINLINE DRIVES, HALF SIZE	$659.95
COLOR COMPUTER II 64K W/EXT. BASIC	$189.95

CONTROLLERS

J&M DISK CONTROLLER W/ JDOS OR RADIO SHACK DISK BASIC, SPECIFY WHAT DISK BASIC.	$139.95
RADIO SHACK DISK CONTROLLER 1.1	$134.95

DISK DRIVE CABLES

CABLE FOR ONE DRIVE	$ 19.95
CABLE FOR TWO DRIVES	$ 24.95

MISC

64K UPGRADE W/MOD. INSTRUCTIONS, C,D,E,F, AND COCO 2	$ 49.95
HJL KEYBOARDS	$ 69.95
MICRO TECH LOWER CASE ROM ADAPTER	$ 74.95
RADIO SHACK BASIC 1.2	$ 29.95
RADIO SHACK DISK BASIC 1.1	$ 29.95
RADIO SHACK EXT. BASIC	$ 39.95
SCREEN CLEAN CLEARS UP VIDEO DISTORTION	$ 39.95
MEMOREX DISKS 5" SS,DD	$ 24.00
SHIPPING INCLUDED ON DISK PRICES	
DISK DRIVE CABINET & POWER SUPPLY	$ 49.95
SINGLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$199.95
DOUBLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$249.95

PRINTERS

EPSON RX-80	$329.00
EPSON RX-80FT	$379.00
EPSON MX-100	$650.00
EPSON FX-100	$799.00
EPSON FX-80	$549.00
EPSON MX-70	$200.00

SERIAL BOARDS FOR PRINTERS

MX-SERIES	$119.95
FX-SERIES	$ 99.95

USA ADD 2% SHIPPING
FOREIGN ADD 5% SHIPPING

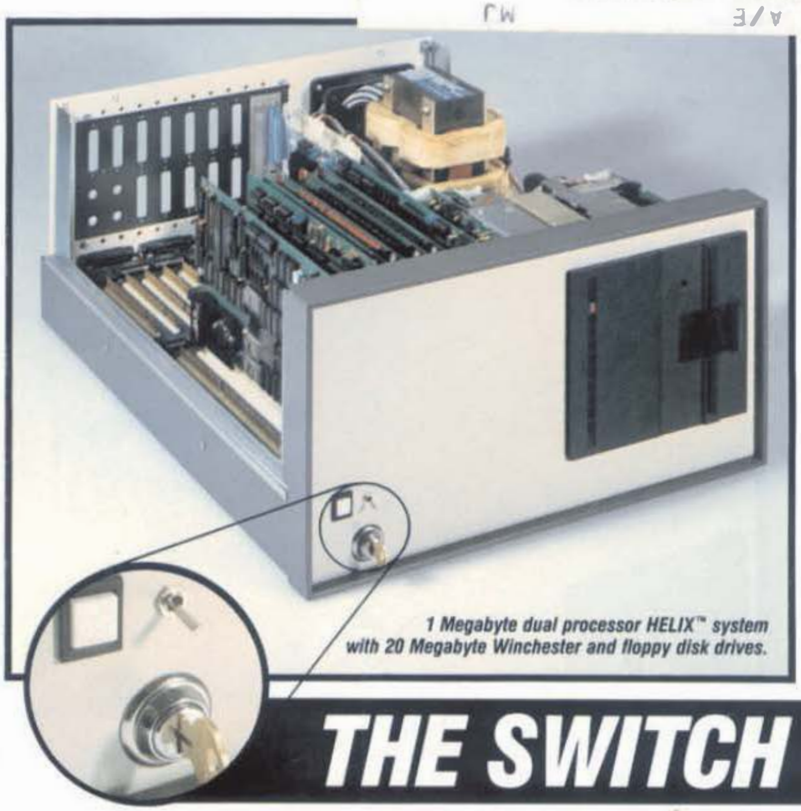
SPECIAL MX-100 $550.00

5900 Cassandra Smith Rd. Hixson, TN 37343

*FLEX is a Trademark of Technical System Consultants
*STAR-DOS+ is a Trademark of STAR-Kits & Data-Comp

68 Micro Journal
ISSN 0194-5025
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Hixson, TN
And Additional Mailing Offices



1 Megabyte dual processor HELIX™ system with 20 Megabyte Winchester and floppy disk drives.

[illegible] AIN HAZELWOOD COMPUTER SYSTEMS demonstrates its leadership in computer technology by delivering the only computer system capable of switching between either the 6809 or the 68000 processor. Switching is easily accomplished by a simple front panel toggle switch. The reason we can offer this exclusive feature now, is that when our proven 6809 processor board was designed several years ago, we had the foresight to include the bus controls that allow processor switching.

Hazelwood Computer Systems is also proud to be the first S-50/S-64 bus manufacturer to license and deliver the OS9/68K Operating System from Microware Systems Corporation. OS9/68K is the 88000 version of the popular and powerful OS9 Operating System. Utilizing our proven MC-20 disk controller, OS9/68K can conveniently share a Winchester disk with OS9. Changing from 8809 to 68000 operation is as simple as switching processors and booting the new system from the Winchester disk.

The ease of switching processors and operating systems makes a HELIX™ dual processor system the natural choice for software development. In addition, the advanced design of HELIX™ equipment, emphasizing performance and reliability, makes HELIX™ boards and systems the best value in computing offered anywhere.

System prices vary with configuration. Call for exact pricing.

THE SWITCH IS ON...

The CP-08 processor board utilizes a 68008 processor running at 10 Mhz clock rate. Using proprietary bus synchronization circuitry and single cycle DMA, the CP-08 achieves a marked performance increase over a 2 MHz 6809. Offering absolute compatibility with the 68000 instruction set, the 68008 addresses up to 1 Megabyte of memory. Also included on the CP-08 are up to 4K of ROM, an interrupt timer, and, with battery backup operation, a clock/calendar and 2K RAM. Implemented as a standard S-50 board, the CP-08 brings 68000 operation to S-50 bus computers.

ORDER: CP-08 PRICE: $595

The MC-20 Mass Storage Controller board interfaces up to 4 floppy and 8 Winchester disk drives to the S-50/S-64 bus. The MC-20 is an intelligent controller with its own 2 Mhz 6809 processor and 56K RAM. It provides DMA data transfers to a full 24 bit address. All disk operation requests are by logical block number, with the controller performing the necessary track/sector address calculations. Any combination of 5¼ or 8 inch floppy drives can be accommodated with all drive parameters, such as write precompensation, software controlled for each individual drive. Winchester drives are connected via a SASI bus interface. Block address mapping is provided which allows a single drive to be segmented into several logical units. The MC-20 is the controller of the MS-20 Mass Storage Subsystem which includes a 20 Megabyte Winchester drive.

ORDER: MC-20 PRICE: $695

OS9/68K

OS9/68K offers increased performance and larger user memory space while retaining all of the features of OS9. Disk file compatibility and operational similarity assures that present OS9 users can easily transfer their operations to the 68000. Included are an editor, assembler, linker, and debugger. A C compiler is available now. BASIC09 and other languages will be avilable soon.

ORDER: OS9/68K PRICE: $250

All items available stock to 30 days.
Prices subject to change without notice.

HAZELWOOD COMPUTER SYSTEMS

907 East Terra, O'Fallon, MO 63366, 314-281-1055

HELIX™

OS9 and OS9/68K are registered trademarks of microware Systems Corp. HELIX is a trademark of Hazelwood Computer Systems.